Problem 1: Twenty Questions

In the game of "twenty questions" I think of an item (like a fish) from a set of N items, and you get to ask me at most twenty questions that can only be answered "yes" or "no" to identify the item. For example, you might ask "Is it living?" If I answer "yes," then you might ask, "Does it have fur?" If I answer "no," then you might then ask, "Does it have fins?" This continues until you either guess the item (in which case you win), or you've asked twenty questions without identifying the item (in which case I win).

With just 20 questions you could identify any one of 524,288 items, assuming you can distinguish among them by asking 19 "yes/no" questions and then with your 20th question ask, "Is it X?" Of course it might take fewer than 20 questions if you have a good idea about the identity of the item, but in this problem we'll assume all the questions are used.

Suppose you could ask questions that could be answered with more than just a "yes" or "no." For example, suppose you could ask, "Does it weight less than, equal to, or greater than 10 pounds?" This question has three possible answers. Then how many questions would you need to ask in order to distinguish among N items?

In this problem you will be told how many different answers can be given to each of your questions, and the number of items in the set of possible choices for the item. All you need to do is determine the maximum number of questions that must be asked to identify the item. This assumes that your questions are chosen in such a way as to divide the remaining candidates for the item into suitably sized groups.

Input

There will be multiple cases to consider. For each case there will be two integers in the input. The first integer, K, is the number of possible answers to each question (no larger than 10). The second integer, N, is the number of items in the set of possible choices (no larger than 2,147,483,647). A pair of zeroes will follow the last case.

Output

For each input case, display a single line that looks like this:

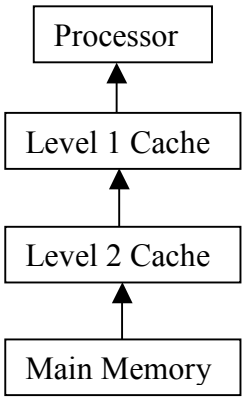
***N* items, *K* answers per question, *M* questions**

where N and K are the values from the input, and M is the maximum number of questions required.

Sample Input	Expected Output
2 524288	524288 items, 2 answers per question, 20 questions
3 524288	524288 items, 3 answers per question, 13 questions
4 524288	524288 items, 4 answers per question, 11 questions
3 9	9 items, 3 answers per question, 3 questions
10 1000	1000 items, 10 answers per question, 4 questions
0 0	

Problem 2: Cache Simulation

Most modern computers use one or more levels of cache memory between the processor and the main memory to minimize the time the processor has to wait for information from main memory. Each cache level is characterized as having some number of memory blocks, each of which has a fixed size (measured in bytes, and always a power of 2); the total size of a cache level is just the number of blocks in that cache level times the size of a block. The address of the lowest-numbered byte in each block is always a integral multiple of the block size, and the bytes in a block have contiguous addresses. For example, with a block size of 16, the bytes in a block might possibly be numbered 16 though 31, or 32 through 47, or 160 through 175.



The processor in this problem only reads single bytes, and it does so by issuing a request that specifies the address of the desired byte. If the byte is in the cache closest to the processor (known as the "level 1 cache"), then that cache delivers the byte to the processor; the length of time required for this operation is called the "level 1 access time." If the byte desired by the processor is not in the level 1 cache, but is in the level 2 cache (which is just below the level 1 cache), then the (level 1 size) block containing that byte is delivered from the level 2 cache to the level 1 cache, and then the desired byte is delivered from the level 1 cache to the processor. The total time required in this case is the time required by the level 2 cache to deliver the block to the level 1 cache (naturally called the "level 2 access time"), plus the level 1 access time for the single byte. This pattern continues through all lower cache levels (if present) to the main memory, if necessary. Thus, if a byte requested by the processor isn't in any of the cache levels, the total access time required is the sum of the access times of each cache level plus that of the main memory. The figure to the left illustrates the flow of information in a system with two cache levels.

Each cache is initially empty. When a block is retrieved from a lower-level cache or main memory, it is placed in an empty block in the cache. When no empty blocks are available, and a new block is requested, it will replace an existing block. The particular block it replaces is that block that has been least recently used.

In this problem you will be given the number of caches in a system (between 1 and 3), the block size and total size of each cache, and the access time for each cache and the main memory. Times will be in integral numbers of nanoseconds (nsec). You will then be given a list of the addresses of bytes requested by the processor, and are to compute the time the processor must wait for all of the bytes to be delivered.

As a simple (but unreaslistic) example, suppose the system has two caches. The level 1 cache has 16 byte blocks, a total size of 32 bytes (that is, 2 blocks), and an access time of 4 nanoseconds. The level 2 cache has 32 byte blocks, a total size of 64 bytes (2 blocks), and an access time of 10 nanoseconds. Main memory has an access time of 50 nanoseconds. Suppose the processor requests, in order, bytes from locations 10, 20, 30, 40, and 50. (Cache blocks are numbered here for reference.)

- Since both levels are initially empty, 32 bytes from main memory locations 0 through 31 will be placed in level 2 block 0 (50 nsec), then 16 bytes from that block (addresses 0 through 15) will be placed in level 1 block 0 (10 nsec). Finally, the byte with address 10 will be delivered to the processor (4 nsec). Total time to access the first byte is 64 nsec.
- The byte with address 20 isn't found in level 1, but is found in level 2 in block 0. The 16 bytes containing address 20 (16 to 31) are placed in level 1 block 1 (10 nsec), and the byte with address 20 is delivered to the processor (4 nsec), for a total time of 14 nsec. Note that both blocks in the level 1 cache are now in used.

- Next, the byte with address 30 is sought. Since it is found in the level 1 cache, that byte is simply delivered to the processor (4 nsec).
- Now address 40 is issued by the processor. The byte at this location is not in either cache level, so the corresponding 32 byte block (addresses 32 to 63) is delivered to the level 2 cache and placed there in block 1 (which was previously unused), taking 50 nsec. Then the 16 bytes containing address 40 (32 to 47) are delivered to the level 1 cache (10 nsec more). These 16 bytes are placed in block 0 of the level 1 cache, since it is the least recently used (block 1 of the level 1 cache was used to satisfy the processor's request for address 30). Finally, the byte is delivered to the processor, for a total time of 64 nsec.
- Finally address 50 is requested. Found in the level 2 cache in block 1, the appropriate 16 bytes (48 to 63) are delivered to the level 1 cache (10 nsec). These bytes are placed in block 1 of the level 1 cache, since block 0 was just used. The selected byte is delivered to the processor (4 nsec), for a total time requirement of 14 nsec.

The total time for the bytes at this sequence of addresses to be delivered to the processor is 64 + 14 + 4 + 64 + 14 = 160 nsec.

Input

There will be multiple input cases. For each case, the input begins with an integer that specifies the number of cache levels (between 1 and 3). For each cache level, starting with level 1, the input then contains integers giving the block size, total size, and access time for the cache level. Each cache level has mo more than 100 blocks, and a block size that is no larger than the next (lower level) cache. Next there appears an integer giving the access time for the main memory. Finally, there appears an integer specifying the number of addresses requested by the processor (no more than 1,000) followed by those addresses in the order they were requested; each address is in the range 0 to 65535. A single 0 follows the input for the last case.

Output

For each case, display a single line containing the case number (1, 2, …) and the total time required for all of the bytes requested by the processor to be delivered.

Sample Input

```
2
 16 32 4
 32 64 10
 50
 5 10 20 30 40 50

2
 8 48 4
 32 64 10
 50
 5 10 20 30 40 50

0
```

Expected Output

```
Case 1: total time = 160 nanoseconds
Case 2: total time = 170 nanoseconds
```

Problem 3: No Priority

A fully-parenthesized expression has, as you might expect, parentheses surrounding every pair of operands of a binary operator. Consider a language where all binary operators have the same precedence so that parentheses are required to guarantee a particular order of evaluation. For example, an expression such as

```
1+2-3*4-5
```

can be parenthesized in 14 ways:

```
(((1+2)-3)*4)-5)
(((1+(2-3))*4)-5)
(((1+2)-(3*4))-5)
(((1+2)-3)*(4-5))
((1+((2-3)*4))-5)
((1+(2-(3*4)))-5)
((1+(2-3))*(4-5))
((1+2)-((3*4)-5))
((1+2)-(3*(4-5)))
(1+(((2-3)*4)-5))
(1+((2-(3*4))-5))
(1+((2-3)*(4-5)))
(1+(2-((3*4)-5)))
(1+(2-(3*(4-5))))
```

Assume that each expression utilizes only integer constants consisting of one or more decimal digits, and that each binary operator is represented by one of the characters `'+'`, `'-'`, `'*'` or `'/'` (without the quotes, of course). Write a program that will read expressions, each contained on one input line and then generate all possible fully-parenthesized expressions for each one. No duplicates are allowed.

Input

The input will contain multiple cases. The input for each case will be an unparenthesized expression of the form described above, immediately followed by an end of line character. That is, each input line will contain an integer constant, followed by zero or more occurrences of an operator and an integer constant, followed by the end of line character. Input for the last case will be followed by a line containing only the end of line character.

Output

For each case, first display the case number (they start with 1 and increase sequentially) and the number of unique fully parenthesized expressions, both on the same line. Follow this line with additional lines containing the parenthesized expressions, one per line, each indented by four spaces from the beginning of the line. The expressions may appear in any order. No output expression will contain more than 128 characters.

Sample Input	Output for the Sample Input

`1+2*3`
`1/2/3/4`
line with only end of line character
	`Case 1: 2 expressions`
	`    ((1+2)*3)`
	`    (1+(2*3))`
	`Case 2: 5 expressions`
	`    (((1/2)/3)/4)`
	`    ((1/2)/(3/4))`
	`    ((1/(2/3))/4)`
	`    (1/((2/3)/4))`
	`    (1/(2/(3/4)))`

Problem 4: The Fence Builder

A fence builder has been given a strange task. Provided with N (between 3 and 100) pieces of straight fencing, each having an arbitrary length, the builder is to enclose as large a region as possible. The customer wants to know the area of the region that can be enclosed by the fence before it is built. There is only one constraint on the construction: each piece of fencing is connected only at its endpoints to exactly two other different pieces of fencing. That is, after completion, the fence will look like a (possibly irregular) polygon with N sides. The customer has guaranteed the builder that the fencing provided will allow for a region with a non-zero area to be enclosed.

Input

There will be multiple cases in the input. For each case, the input begins with the number of pieces of fencing (an integer, N). There then follow N positive, non-zero real numbers giving the lengths of the fence pieces. A single integer zero follows the last case in the input.

Output

For each case, display the case number (starting with 1) and the maximum area that can be enclosed by the provided fencing materials. Show three fractional digits in each answer. Use the format shown below in displaying the results.

Sample Input

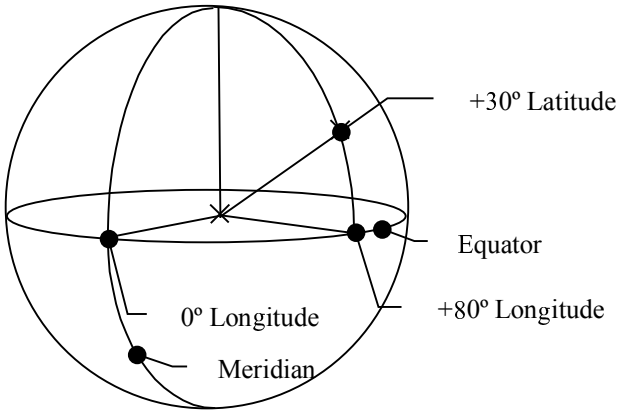
```
3 2.0 2.0 2.0
4 1.0 1.0 1.0 1.0
4 5.0 5.0 3.0 11.0
0
```

Expected Output

```
Case 1: maximum area = 1.732
Case 2: maximum area = 1.000
Case 3: maximum area = 21.000
```

Problem 5: Communication Planning for Phobos

Life has been found on Phobos, one of the satellites of Mars! Unfortunately, the life forms there aren't quite as advanced as those on Earth, and they don't have modern communications (at least by Earth standards). The Advanced Communication Management Company (ACM) has decided to build a central office and connect the Phobosians' homes for communication (telephone, television, Internet, and so forth). They naturally want to minimize their capital outlay in this effort, and they need to decide how to lay fiber optic cable (essentially on the surface) so the smallest amount is used. Since ACM uses digital broadband technology, it is only necessary that there be a cable path that connects every subscriber and the central office. That is, there does not necessarily need to be a separate cable from the central office to each subscriber's home.



We know the precise location of each Phobosian's home and the planned ACM central office on the surface. These are given using longitude and latitude. Longitude is measured from an arbitrary meridian on the surface of Phobos, and has values in the range −180 degrees to +180 degrees. Latitude is measured from the equator, and has values in the range −90 degrees to +90 degrees. For planning purposes we assume Phobos is perfectly spherical, exactly 16.7 miles in diameter. The figure to the left illustrates one possible location (+80° longitude, +30° latitude).

INPUT

There will be one or more sets of input data. Each set will contain, in order, an integer N no larger than 100, but at least 2, followed by N pairs of real numbers, each pair giving the unique longitude and latitude, in degrees, of a Phobosian's home or the central office. A single integer zero will follow the last data set.

OUTPUT

For each input data set print a single line containing the data set number (1, 2, …) and the number of miles of cable required to connect all the Phobosian's homes and the central office; show two fractional digits in the distance.

SAMPLE INPUT

```
3
0 0    0 90    0 -90

3
0 0    0 90    90 0

3
0 0    90 0    45 0

6
-10 10   -10 -10   0 0   90 0   80 20   100 -10

0
```

EXPECTED OUTPUT

```
Case 1: 26.23 miles
Case 2: 26.23 miles
Case 3: 13.12 miles
Case 4: 21.16 miles
```

Problem 6: Clustering

A psychologist tests N different individuals. The score for each individual is a pair of real numbers, x and y, that the psychologist treats as points in a plane. The psychologist then wants to separate the individuals into at least 2 groups, with at least 2 individuals in each group, based on the proximity of their scores. The decision is made to use the Euclidian distance between two scores as a measure of their proximity. That is, for two scores (x_1,y_1) and (x_2,y_2), their proximity is defined by the value of the expression

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} .$$

To place the individuals in groups, the psychologist decides to use the following steps. (1) The two individuals whose scores are closest together are placed in group 1; call them A and B. (2) The next two individuals (not in group 1) with the closest scores are placed in group 2; call them C and D. (Clearly at least four individuals must be tested.) (3) The next ungrouped individual, called E, to be included in a group is the one closest to any individual already in a group or any other ungrouped individual. If E is closer to A or B than to C, D, or any other ungrouped individual, then E joins group 1. Likewise, if E is closer to C or D than to A, B, or any other ungrouped individual, then E joins group 2. Finally, if E is closer to another ungrouped individual, say F, than A, B, C, or D, then a new group is formed from E and F. If the shortest distance to an existing group is within 0.001 of the shortest distance to an ungrouped individual, or doesn't uniquely identify the group in which E should be placed, then E is placed in the existing group created earliest that has a member closest to E. This step (3) is repeated for all remaining ungrouped individuals.

Input

There will be multiple cases. For each case there appears an integer N specifying the number of individuals tested; N will be at least 4, but no larger than 100. There will then follow N pairs of real x and y values, one for each individual. The first pair is for individual 1, the second pair is for individual 2, and so forth.

Output

For each case, first display the case number (they're numbered sequentially starting with 1) on a line by itself. Then display the identity of the individuals in each group, one line per group. Use the format shown in the samples below. Specifically, output lines must be no longer than 79 characters, indentation (as shown in the samples) is required, and commas are required between the individual identity numbers.

Sample Input

```
4
  0.0 0.0 0.0 3.0 4.0 0.0 3.0 5.0
6
  0.0 0.0 0.0 3.0 4.0 0.0 3.0 5.0 2.0 6.0 3.0 1.0
25
  1 0 1 1 100 100 100 101 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 9
  1 10 1 11 1 12 1 13 1 14 1 15 1 16 1 17 1 18 1 19 1 20 1 21 1 22
0
```

Expected Output

```
Case 1:
  Group 1: 1, 2
  Group 2: 3, 4
Case 2:
  Group 1: 3, 6
  Group 2: 4, 5
  Group 3: 1, 2
Case 3:
  Group 1: 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
           21, 22, 23, 24, 25
  Group 2: 3, 4
```

Problem 7: Off Base

Integers in bases larger than 10 are usually represented using letters to represent the digits which have values larger than 9. For example, hexadecimal (base 16) integers are written using the digits 0 through 9 (to represent themselves) and the letters A through F to represent the digits with values 10 through 15, respectively. In a similar manner, the remaining letters in the alphabet could be used to represent the digits with values 16 through 35. This allows for easy display of values in any base from 2 through 36.

In rummaging through a collection of files on an old reel of magnetic tape, a computer archeologist came across a file containing a sequence of what appeared to be arithmetic formulas. The expressions were of the form

number-1 operator number-2 = number-3

where number-1, number-2, and number-3 are formed from the digits 0-9 and the letters A-Z, and operator is one of '+' or '-' – presumably meaning addition or subtraction. The archeologist would like to know if these expressions really do represent valid expressions, and if so, in what base the numbers were written. The assumption is made that the "digits" A through Z (upper case only) do represent digits with values 10 through 35, none of the numbers are negative, and none of the numbers contain more than 50 digits.

You volunteer to help. At the outset, you know that examining a single number is insufficient to determine its base. For example, the number 77 could be written using any base greater than 7. If, however, you should see the expression

77 + 22 = 99

then you can easily tell than 10 is the smallest base that could have been used. On the other hand, if the expression was

77 + 22 = 121

then you can determine that the numbers were written in base 8. Your problem is to write a program that will assist the archeologist in determining the smallest base used to represent the numbers in each expression, and to also identify those expressions that couldn't have been expressed using a number base in the range 2 to 36.

Input

There will be multiple cases. For each case there will be a single line containing an expression. Blanks may appear before or after any of the numbers, the operator, or the equal sign, but they are not required. An empty line (that is, one containing only zero or more blanks) will follow the expression for the last case.

Output

For each case, display the case number (starting with 1) and the smallest base that could have been used if the expression is correct. If the expression could not be correct using any of the possible bases, then display the case number and the statement "expression is invalid."

Sample Input

```
77 + 22 = 99
115 + 26 = 143
2K3 - 1A1 = 1J2
2N + M = 3I
```

This line is blank

Expected Output

```
Case 1: minimum base is 10
Case 2: minimum base is 8
Case 3: expression is invalid.
Case 4: minimum base is 27
```

Problem 8: Minimizing Wait Time

When scheduling the use of a resource (like a CPU), one measure of the schedule is the average length of time a client has to wait for the resource. That is, after a client indicates they want to use the resource, and for how long they want to use it, how long must they wait? This clearly depends on how many other clients want to use the resource, and when they are allowed to use it.

Assume that a particular resource is scheduled in integral units of time (for example, milliseconds), and that once a client is granted the use of the resource, they have complete (unshared) use of the resource for the length of time they requested. A schedule is just the order in which a group of clients are allowed to use the resource.

Suppose we know, in advance, the length of time each client wants to use the resource. Your task in this problem is to determine the minimum average time clients must wait before being allowed to use the resource.

For example, suppose we have three clients that want to use the resource. One (client A) wants it for 10 units of time. Another (client B) wants the resource for 13 units of time. Client C wants to use the resource for 6 units of time. If we let client A use the resource first, it waits 0 units of time. If we let B use it second (after A), then B waited a total of 10 units of time – exactly the length of time the resource was used by client A. Finally, client C uses the resource after waiting 10 units of time for A to finish and another 13 units of time for B to finish. So C had to wait 23 units of time. The average wait time is (0 + 10 + 23) / 3, or 11 units of time. As it turns out, the best schedule would require an average wait time of only 7 1/3 units of time.

Input

The input will contain multiple cases. The number of cases is specified by the first integer in the input. The input for each case will consist of an unknown number of requests for the use of the resource. Each of these is represented by a positive integer separated from the others by whitespace. The last request in each case is followed by a negative number.

Output

For each case, display the case number (they start with 1 and increase sequentially), the number of requests for the resource, and the minimum average time clients had to wait for the resource. Display this number with three fractional digits. Your output should resemble that shown in the sample below.

Sample Input

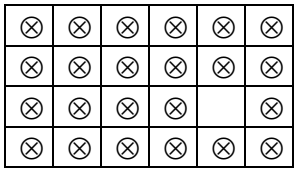
```
3 10 13 6 -1 54 95 10 57 -1 20
23 6 7 18 34 17 92 4 75 90 24 94
47 68 86 46 29 68 77 28 94 82 16
47 40 44 61 31 55 89 53 95 22 61 -1
```

Output for the Sample Input

```
Case 1: 3 requests scheduled. Average wait: 7.333
Case 2: 4 requests scheduled. Average wait: 48.750
Case 3: 35 requests scheduled. Average wait: 555.714
```

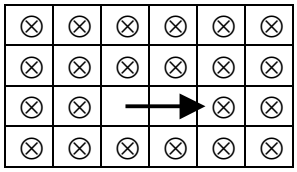
Problem 9: Coke Bottles

A long time ago – before aluminum cans and plastic bottles, Coke used to actually come in glass bottles. A case of Coke consisted of a box or carton that had 24 bottles, arranged as 4 rows of 6 bottles each. Unfortunately in this box one bottle is missing:

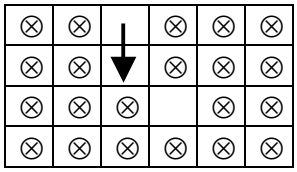


In this instance the missing bottle comes from row 3 column 5; the rows are numbered from 1 to 4, and the columns are numbered from 1 to 6.

We will play a game based on this diagram. You can "jump over" a bottle, either horizontally or vertically, and remove the bottle you have jumped over. That is, you can move a bottle over another bottle to an empty space in the box, and then remove the bottle over which you jumped. For example, I can take the bottle in row 3 column 3, jump over the one in row 3 column 4, and land in row 3, column 5. Since I can do this, I can remove the bottle from row 3 column 4:



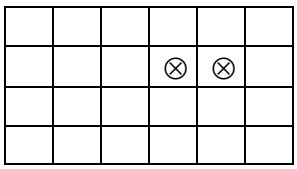
Now the next move from here might be to jump from row 1 column 3, over row 2 column 3, and into row 3 column 3, removing the bottle in row 2 column 3:



Possible moves from here include:

[3,2] to [3,4] removing [3,3] (moving right from [3,2])
[4,3] to [2,3] removing [3,3] (moving up from [4,3])
[3,6] to [3,4] removing [3,5] (moving left from [3,6])
etc.

Can you eventually get to a point where there is only one bottle left? The very last move might be something like this, where we can move from row 2 column 5 over row 2 column 4, and into row 2 column 3 (left from [2,5]):



In this problem you are given the description of a hypothetical Coke case containing fewer than 10 rows and 10 columns, the position (row and column) of a missing bottle, and the maximum number of bottles that can be left. Your job is to determine a sequence of "jumps" that will achieve this task. The valid moves are to jump up and over, down and over, left and over, or right and over. Diagonal jumps are not allowed; you may not drink the Cokes to create empty spaces.

Input

The input will contain multiple cases. For each case the input will contain five integers. The first two integers specify the number of rows and columns of the Coke case; neither of these will be larger than 9 or smaller than 1. The second two integers specify the row and column where a bottle is initially missing. The last integer specifies the maximum number of bottles that may be left. The last input case will be followed by five zeroes.

Output

For each case, first display the case number (they start with 1 and increase sequentially). If there is no solution for the case, then display the message "No solution". Otherwise, starting on the next line, display the sequence of "jumps" that will leave just the specified number of bottles in the case. Each jump is specified in the form (r,c,dir), where r and c specify the row and column number of the bottle that jumps (NOT the jumped bottle), and dir is the direction that the bottle moved (U, D, L, or R for up, down, left, or right). Display these in the style shown in the sample output below, with as many jumps per line as possible, but with no more than 80 characters per output line, including four blank characters of indentation at the beginning of each line. Separate the output for consecutive cases with a blank line. If there are multiple valid jump sequences, any of them is acceptable.

Sample Input

```
4 6 1 1 1
1 3 1 1 1
3 3 2 2 1
3 3 1 1 4
0 0 0 0 0
```

Output for the Sample Input

```
Case 1:
    (3,1,U), (3,3,L), (4,1,U), (4,3,L), (4,5,L), (3,5,L), (1,5,D), (3,6,L),
    (3,4,L), (2,3,R), (1,6,D), (4,6,U), (2,6,L), (1,4,D), (1,1,D), (4,1,U),
    (1,3,L), (1,1,D), (3,1,R), (4,3,U), (2,2,R), (3,4,U)

Case 2:
    (1,3,L)

Case 3: No solution

Case 4:
    (3,1,U), (3,3,L), (1,3,D), (1,2,D)
```