Problem Set of Practice Contest

Japanese Alumni Group

Contest Held on: 30 Oct 2005

Contents

The problem set consists of nine problems identified by A through I.

Problems by Japanese Alumni Group

Problems A, B, D, E, G and I were newly created for Practice Contest by the members of Japanese Alumni Group.

You may use these problems in any form, entirely or in part, with or without modification, for any purposes, without prior or posterior consent to Japanese Alumni Group, provided that your use is made solely at your own risk.

Problems from Past Contests

Problems C, F and H were taken from other programming contests as listed below:

Problem	Source
Problem C	ACM ICPC Arab and North Africa 2004
Problem F	ACM ICPC Guangdong Collegiate Programming Contest 2005
Problem H	ACM ICPC Singapore 2001

These problems were modified by the members of Japanese Alumni Group for clarification of the statement, adjustment of the difficulty level in solving the problems, attempt to hide the origin of the problems from the contestants during the contest proper, and other purposes. The origin of each problem, however, was revealed to the contestants at the review just after the contest proper.

Please note that permission to use these problems *cannot* be granted by Japanese Alumni Group, since they are not originated with us.

Disclaimer

Problem A
Blackjack
Input: A.txt

Brian Jones is an undergraduate student at Advanced College of Metropolitan. Recently he was given an assignment in his class of Computer Science to write a program that plays a dealer of blackjack.

Blackjack is a game played with one or more decks of playing cards. The objective of each player is to have the score of the hand close to 21 without going over 21. The score is the total points of the cards in the hand. Cards from 2 to 10 are worth their face value. Face cards (jack, queen and king) are worth 10 points. An ace counts as 11 or 1 such a way the score gets closer to but does not exceed 21. A hand of more than 21 points is called *bust*, which makes a player automatically lose. A hand of 21 points with exactly two cards, that is, a pair of an ace and a ten-point card (a face card or a ten) is called a *blackjack* and treated as a special hand.

The game is played as follows. The dealer first deals two cards each to all players and the dealer himself. In case the dealer gets a blackjack, the dealer wins and the game ends immediately. In other cases, players that have blackjacks win automatically. The remaining players make their turns one by one. Each player decides to take another card (*hit*) or to stop taking (*stand*) in his turn. He may repeatedly hit until he chooses to stand or he busts, in which case his turn is over and the next player begins his play. After all players finish their plays, the dealer plays according to the following rules:

- Hits if the score of the hand is 16 or less.
- Also hits if the score of the hand is 17 and one of aces is counted as 11.
- Stands otherwise.

Players that have unbusted hands of higher points than that of the dealer win. All players that do not bust win in case the dealer busts. It does not matter, however, whether players win or lose, since the subject of the assignment is just to simulate a dealer.

By the way, Brian is not good at programming, thus the assignment is a very hard task for him. So he calls you for help, as you are a good programmer.

Your task is to write a program that counts the score of the dealer's hand after his play for each given sequence of cards.

Input

The first line of the input contains a single positive integer N, which represents the number of test cases. Then N test cases follow.

Each case consists of two lines. The first line contains two characters, which indicate the cards in the dealer's initial hand. The second line contains eight characters, which indicate the top eight cards in the pile after all players finish their plays.

A character that represents a card is one of A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q and K, where A is an ace, T a ten, J a jack, Q a queen and K a king.

Characters in a line are delimited by a single space.

The same cards may appear up to four times in one test case. Note that, under this condition, the dealer never needs to hit more than eight times as long as he or she plays according to the rule described above.

Output

For each case, print on a line "`blackjack`" if the dealer has a blackjack; "`bust`" if the dealer busts; the score of the dealer's hand otherwise.

Sample Input

```
4
5 4
9 2 8 3 7 4 6 5
A J
K Q J T 9 8 7 6
T 4
7 J A 6 Q T K 7
2 2
2 3 4 K 2 3 4 K
```

Output for the Sample Input

```
18
blackjack
21
bust
```

Problem B
Eight Princes
Input: B.txt

Once upon a time in a kingdom far, far away, there lived eight princes. Sadly they were on very bad terms so they began to quarrel every time they met.

One day, the princes needed to seat at the same round table as a party was held. Since they were always in bad mood, a quarrel would begin whenever:

- A prince took the seat next to another prince.
- A prince took the seat opposite to that of another prince (this happens only when the table has an even number of seats), since they would give malignant looks each other.

Therefore the seat each prince would seat was needed to be carefully determined in order to avoid their quarrels. You are required to, given the number of the seats, count the number of ways to have all eight princes seat in peace.

Input

Each line in the input contains single integer value N, which represents the number of seats on the round table. A single zero terminates the input.

Output

For each input N, output the number of possible ways to have all princes take their seat without causing any quarrels. Mirrored or rotated placements must be counted as different.

You may assume that the output value does not exceed 10^{14}.

Sample Input

```
8
16
17
0
```

Output for the Sample Input

```
0
0
685440
```

Problem C
Geometry?! Why Not??
Input: C.txt

Consider a 2D path drawn in the following manner: Starting at the origin point, we can move only up or right. The path will be described as a string made of zero or more {'U', 'R'} letters. For each 'U' we'll move one unit up, while 'R' moves one unit to the right. In the following figure, the path constructed by the string RRRURRUUURRRRRUUR is drawn in a thick line.

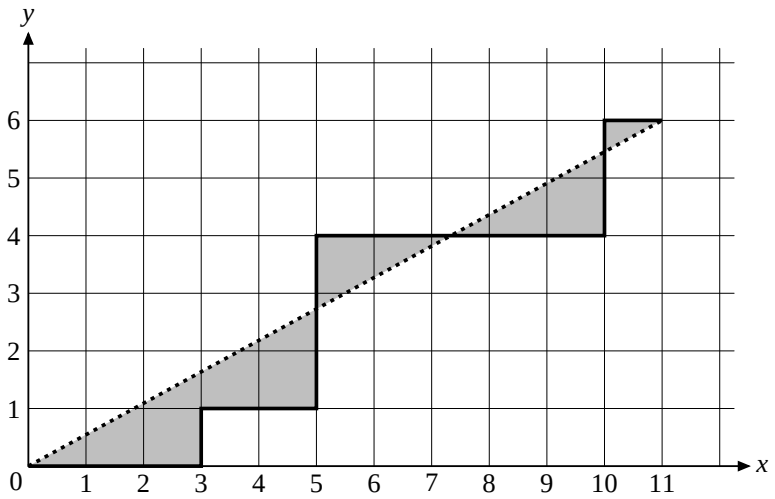


Figure 1: Sample 2D Path

Imagine now that we draw a straight line that connects the origin point to the last point in the path (the line that is drawn in dots in the figure above). We want to compute the total area that falls between the straight line and the path (the grayed area in the above figure).

Input

The input consists of one or more test cases. Each case is described on a separate line. The path of each test case is described as a string made of one or more letters, each of which is either 'U' or 'R', followed by the letter 'S'.

All paths in the input can be drawn on a grid of size 1000×1000.

The last line of the input is made of a single 'S' character and is not part of the test cases.

Output

For each case, write the area on a line. The area may have an arbitrary number of decimal digits, but may not contain an error greater than 10^{-3}.

Sample Input

```
RRRURRUUURRRRRUURS
RUURS
S
```

Output for the Sample Input

```
8.515
1.000
```

Problem D
Divisor is the Conqueror
Input: D.txt

Divisor is the Conquerer is a solitaire card game. Although this simple game itself is a great way to pass one's time, you, a programmer, always kill your time by programming. Your task is to write a computer program that automatically solves *Divisor is the Conquerer* games. Here is the rule of this game:

First, you randomly draw N cards ($1 \leq N \leq 52$) from your deck of playing cards. The game is played only with those cards; the other cards will not be used.

Then you repeatedly pick one card to play among those in your hand. You are allowed to choose any card in the initial turn. After that, you are only allowed to pick a card that *conquers* the cards you have already played. A card is said to *conquer* a set of cards when the rank of the card divides the sum of the ranks in the set. For example, the card 7 conquers the set $\{5, 11, 12\}$, while the card 11 does not conquer the set $\{5, 7, 12\}$.

You win the game if you successfully play all N cards you have drawn. You lose if you fails, that is, you faces a situation in which no card in your hand conquers the set of cards you have played.

Input

The input consists of multiple test cases.

Each test case consists of two lines. The first line contains an integer N ($1 \leq N \leq 52$), which represents the number of the cards. The second line contains N integers $c_1, \ldots, c_N$ ($1 \leq c_i \leq 13$), each of which represents the rank of the card. You may assume that there are at most four cards with the same rank in one list.

The input is terminated by a line with a single zero.

Output

For each test case, you should output one line. If there are any winning strategies for the cards of the input, print any one of them as a list of N integers separated by a space.

If there is no winning strategy, print "`No`".

Sample Input

```
5
1 2 3 3 7
4
2 3 3 3
0
```

Output for the Sample Input

```
3 3 1 7 2
No
```

Problem E
Reading a Chord
Input: E.txt

In this problem, you are required to write a program that enumerates all chord names for given tones.

We suppose an ordinary scale that consists of the following 12 tones:

C, $C^\sharp$, D, $D^\sharp$, E, F, $F^\sharp$, G, $G^\sharp$, A, $A^\sharp$, B

Two adjacent tones are different by a *half step*; the right one is higher. Hence, for example, the tone G is higher than the tone E by three half steps. In addition, the tone C is higher than the tone B by a half step. Strictly speaking, the tone C of the next octave follows the tone B, but octaves do not matter in this problem.

A chord consists of two or more different tones, and is called by its *chord name*. A chord may be represented by multiple chord names, but each chord name represents exactly one set of tones.

In general, a chord is represented by a basic chord pattern (*base chord*) and additional tones (*tension*) as needed. In this problem, we consider five basic patterns as listed below, and up to one additional tone.

Chord Name	**Components**
C	C, E, G
C_7	C, E, G, $A^\sharp$
CM_7	C, E, G, B
Cm	C, $D^\sharp$, G
Cm_7	C, $D^\sharp$, G, $A^\sharp$

Figure 2: Base chords (only shown those built on C)

The chords listed above are built on the tone C, and thus their names begin with C. The tone that a chord is built on (the tone C for these chords) is called the *root* of the chord.

A chord specifies its root by an absolute tone, and its other components by tones relative to its root. Thus we can obtain another chord by shifting all components of a chord. For example, the chord name D represents a chord consisting of the tones D, $F^\sharp$ and A, which are the shifted-up tones of C, E and G (which are components of the chord C) by two half steps.

An additional tone, a tension, is represented by a number that may be preceding a plus or minus sign, and designated parentheses after the basic pattern. The figure below denotes which number is used to indicate each tone.

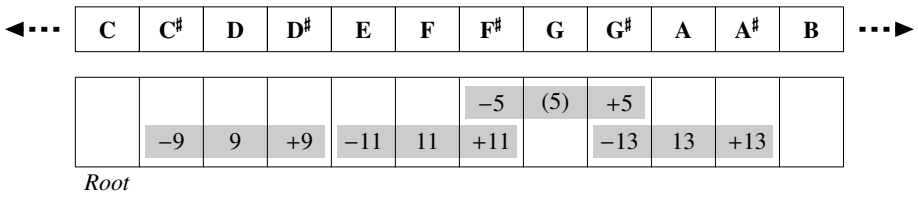


Figure 3: Tensions for C chords

For example, C(9) represents the chord C with the additional tone D, that is, a chord that consists of C, D, E and G. Similarly, C(+11) represents the chord C plus the tone F♯.

The numbers that specify tensions denote relative tones to the roots of chords like the components of the chords. Thus change of the root also changes the tones specified by the number, as illustrated below.

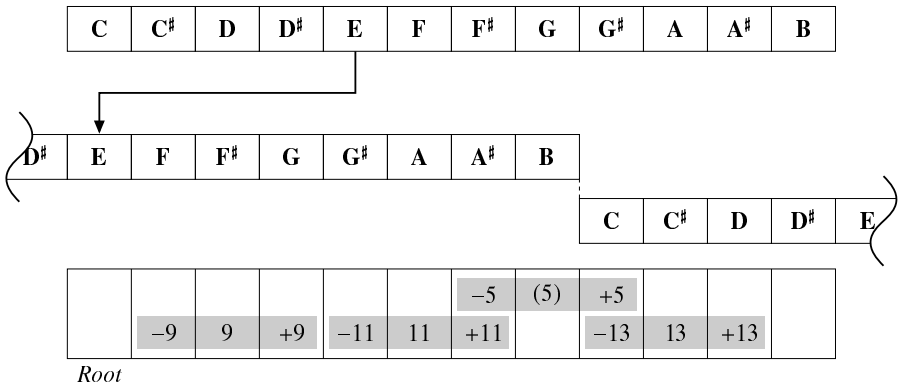


Figure 4: Tensions for E chords

+5 and −5 are the special tensions. They do not indicate to add another tone to the chords, but to sharp (shift half-step up) or flat (shift half-step down) the fifth tone (the tone seven half steps higher than the root). Therefore, for example, C(+5) represents a chord that consists of C, E and G♯, not C, E, G and G♯.

Figure 5 describes the syntax of chords in Backus-Naur Form.

Now suppose we find chord names for the tones C, E and G. First, we easily find the chord C consists of the tones C, E and G by looking up the base chord table shown above. Therefore '`C`' should be printed. We have one more chord name for those tones. The chord Em obviously represents the set of tones E, G and B. Here, if you sharp the tone B, we have the set of tones E, G and C. Such modification can be specified by a tension, +5 in this case. Thus '`Em(+5)`' should also be printed.

```
chord        ::= base_chord tension_part?

base_chord   ::= root_spec ( "" | "7" | "M7" | "m" | "m7" )
root_spec    ::= tone
tone         ::= "C" | "C#" | "D" | "D#" | "E" | "F" | "F#" | "G" | "G#" | "A" |
                 "A#" | "B"

tension_part ::= "(" tension ")"
tension      ::= ( ("+" | "-")? ("9" | "11" | "13") ) | ( ("+" | "-") "5" )
```

Figure 5: Syntax of chords in BNF

Input

The first line of input contains an integer N, which indicates the number of test cases.

Each line after that contains one test case. A test case consists of an integer m ($3 \leq m \leq 5$) followed by m tones. There is exactly one space character between each tones. The same tone does not appear more than once in one test case.

Tones are given as described above.

Output

Your program should output one line for each test case.

It should enumerate all chord names that completely match the set of tones given as input (i.e. the set of tones represented by each chord name must be equal to that of input) in any order.

No chord name should be output more than once. Chord names must be separated by exactly one space character. No extra space is allowed.

If no chord name matches the set of tones, your program should just print '`UNKNOWN`' (note that this word must be written in capital letters).

Sample Input

```
5
3 C E G
3 C E G#
4 C A G E
5 F A C E D
3 C D E
```

Output for the Sample Input

```
C Em(+5)
C(+5) E(+5) G#(+5)
C(13) Am7 Am(+13)
Dm7(9) FM7(13)
UNKNOWN
```

Problem F
University Rankings
Input: F.txt

Doctor Bob is working on creating university rankings at International Company for Promoting Colleges. As we know, a university usually has many different departments, such as department of Computer Science (CS), department of Electronic Engineering (EE), and School of Foreign Languages (FLS). Some of them are quite good when comparing to the other universities, but others are not. So, most of the university rankings are composed of several ranking lists, each list for one department.

But here comes a problem that sometimes it is hard to determine which university is better, when comparing two universities with each other. Fortunately, Bob has advanced a new concept named *absolutely better*, with which the problem above can be partially solved.

Now, we take an example to explain the concept absolutely better:

Assume that there are three universities (X, Y, Z) and every university has three departments: CS, EE and FLS. And the ranking of each department is as follows, where X > Y means X have a better CS department than Y:

- The ranking of CS: X > Y > Z
- The ranking of EE: X > Z > Y
- The ranking of FLS: Z > X > Y

Obviously, all departments of University X are better than those of University Y. Then, we say that X is absolutely better than Y. Using the absolutely better concept, it becomes possible to compare some pairs of the universities.

Now Bob has the complete rankings of different departments of many universities, and he wants to find k universities $(U_1, \ldots, U_k)$ such that U_i is absolutely better that U_j (for any $i < j$). Could you tell Bob the maximum value of k? Note that the universities are represented by numbers in the input.

Input

The first line of the input is a positive integer C, which is the number of test cases followed.

The first line of each test case is two positive integer N, M $(0 < N, M \leq 400)$, where N is the number of universities and M is the number of departments. And then M lines follow. The k-th

($1 \leq k \leq M$) line contains N numbers $U_{k,i}$ ($1 \leq i \leq N$, $1 \leq U_{k,i} \leq N$), indicating the ranking of the k-th department. If University $U_{k,i}$ precedes to University $U_{k,j}$ in line k, then the k-th department of $U_{k,i}$ is better than the k-th department of $U_{k,j}$.

Output

The output should consist of C lines, one line for each test case. Each line only contains one integer — the maximum value of k as described above. No redundant spaces are needed.

Sample Input

```
1
3 3
1 2 3
1 3 2
3 1 2
```

Output for the Sample Input

```
2
```

Problem G
Gather on the Clock
Input: G.txt

There is a self-playing game called *Gather on the Clock*.

At the beginning of a game, a number of cards are placed on a ring. Each card is labeled by a value.

In each step of a game, you pick up any one of the cards on the ring and put it on the next one in clockwise order. You will gain the score by the difference of the two values. You should deal with the two cards as one card in the later steps. You repeat this step until only one card is left on the ring, and the score of the game is the sum of the gained scores.

Your task is to write a program that calculates the maximum score for the given initial state, which specifies the values of cards and their placements on the ring.

The figure shown below is an example, in which the initial state is illustrated on the left, and the subsequent states to the right. The picked cards are 1, 3 and 3, respectively, and the score of this game is 6.

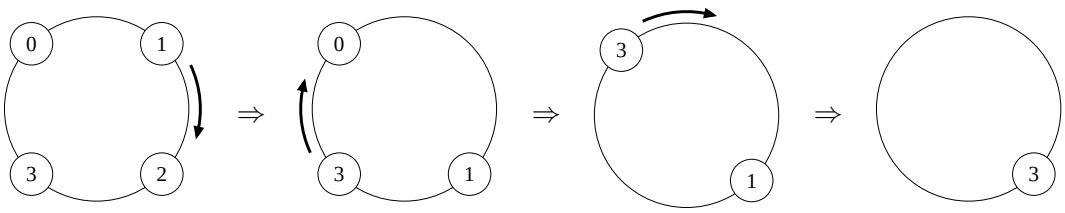


Figure 6: An illustrative play of Gather on the Clock

Input

The input consists of multiple test cases. The first line contains the number of cases. For each case, a line describing a state follows. The line begins with an integer n ($2 \leq n \leq 100$), the number of cards on the ring, and then n numbers describing the values of the cards follow. Note that the values are given in clockwise order. You can assume all these numbers are non-negative and do not exceed 100. They are separated by a single space character.

Output

For each case, output the maximum score in a line.

Sample Input

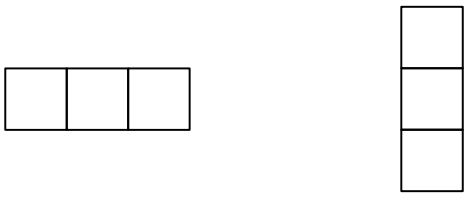
```
2
4 0 1 2 3
6 1 8 0 3 5 9
```

Output for the Sample Input

```
6
34
```

Problem H
Tiling Polygons
Input: H.txt

In a polygon tiling game, you are required to fill a polygon with a set of basic shapes called *tiles*. This is equivalent to cutting up the entire polygon into a number of small pieces; each piece must have exactly the same shape as one of the provided tiles. Let us consider a simple version of the game. Two types of rectangular tiles are given as shown in Figure 7. Their width/height are 1/3 and 3/1 respectively. You are asked to use these two types of tiles to tile a rectilinear polygon (that is, a polygon with only vertical or horizontal edges). For instance, Figure 8(a) shows such a polygon and Figure 8(b) shows how it can be properly tiled with three 3×1 tiles and one 1×3 tile.



(a) (b)

Figure 7: 1×3 tile and 3×1 tile

The coordinate system used in this game is described by the following rules:

1. A polygon is always positioned in such a way that its lowest edge is on the x-axis and its leftmost edge is on the y-axis, as shown in Figure 8.
2. A polygon is represented by an ordered list of points that enumerates all of its vertices. The enumeration starts from the lower-left vertex (i.e. the leftmost vertex lying on x-axis) and proceeds counterclockwise. For instance, the polygon in Figure 8(a) is represented by the sequence of points $[\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{G}, \mathrm{H}]$.
3. A grid (a rectangle whose width and height are both one) is represented by a pair (x, y), where x and y are the x-coordinate and y-coordinate of its top-right vertex. In Figure 8(a), the coordinate of every grid is indicated inside the grid.
4. A tile is designated by a triple (x, y, z), where x and y are the coordinates of the middle grid of the tile and z represents the orientation of the tile. If the tile is horizontal, then $z = 0$; otherwise $z = 1$. Figure 9 gives two examples.

Your task is to write a program that finds a way to tile a given polygon. The polygon is known to be tillable. If there is more than one ways of tiling, you are only required to find one of them.

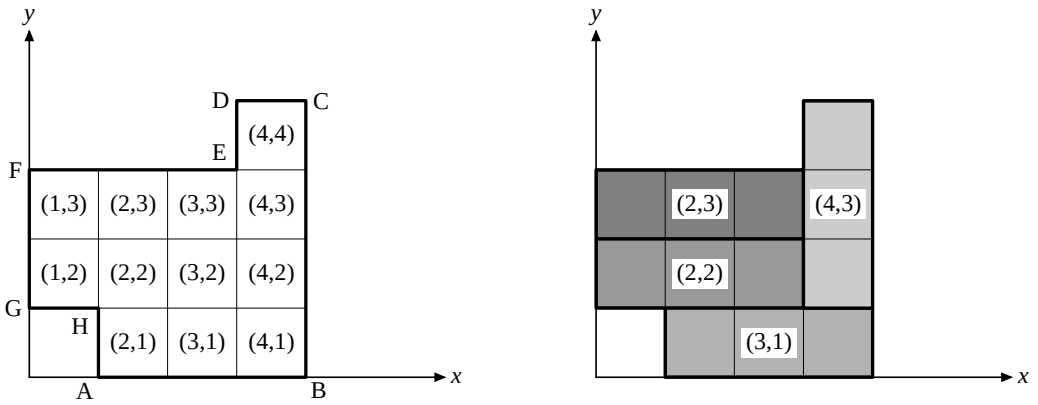


Figure 8: An example of tiling

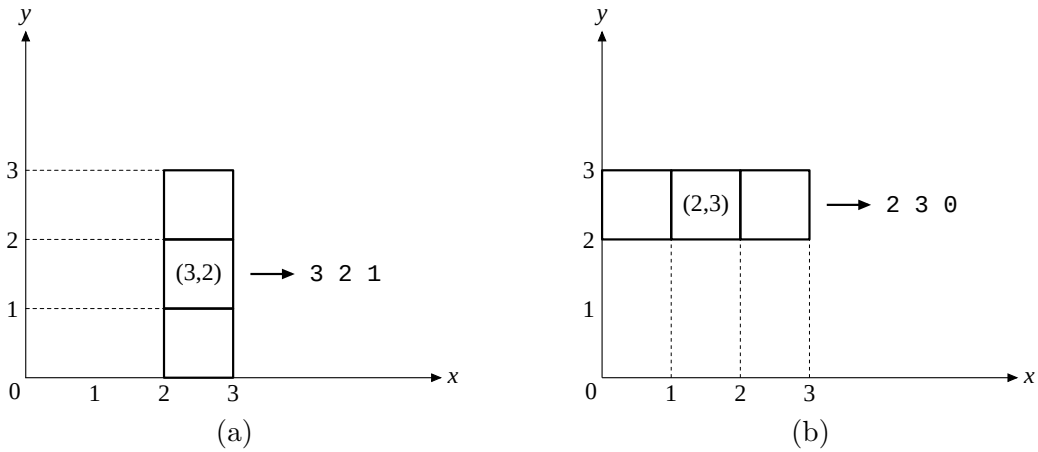


Figure 9: An example of tiling

Input

Input consists of multiple test cases.

Each case describes a polygon to be tiled. The first line contains a single integer N ($4 \leq N < 100$) that denotes the number of vertices of the polygon. The subsequent N lines specify the vertices of the polygon in the order stated above; one line per vertex. A vertex is specified as its x-coordinate followed by its y-coordinate, separated by a white space.

You may assume the area of a polygon is always less than 1000, as well as each coordinate value is an integer less than 100.

A line that contains a single zero indicates the end of input, and this line should not be processed.

The sample input below denotes the polygon shown in Figure 8(a).

Output

For each case, the output produces a list of tiles that fills the polygon specified in the input file. A tile is specified as a triple x y z as described above, separated by a white space; one tile per line. The tiles may appear in any order.

Print a blank line between cases.

The sample output below represents the solution shown in Figure 8(b).

Sample Input

```
8
1 0
4 0
4 4
3 4
3 3
0 3
0 1
1 1
0
```

Output for the Sample Input

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

Problem I
Light The Room
Input: I.txt

You are given plans of rooms of polygonal shapes. The walls of the rooms on the plans are placed parallel to either x-axis or y-axis. In addition, the walls are made of special materials so they reflect light from sources as mirrors do, but only once. In other words, the walls do not reflect light already reflected at another point of the walls.

Now we have each room furnished with one lamp. Walls will be illuminated by the lamp directly or indirectly. However, since the walls reflect the light only once, some part of the walls may not be illuminated.

You are requested to write a program that calculates the total length of *unilluminated* part of the walls.

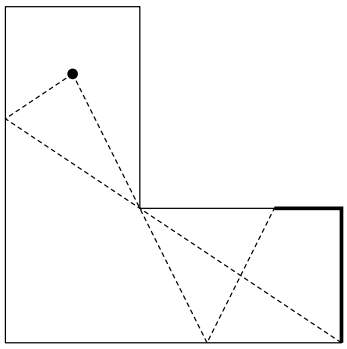


Figure 10: The room given as the second case in Sample Input

Input

The input consists of multiple test cases.

The first line of each case contains a single positive even integer N ($4 \leq N \leq 20$), which indicates the number of the corners. The following N lines describe the corners counterclockwise. The i-th line contains two integers x_i and y_i, where (x_i, y_i) indicates the coordinates of the i-th corner. The last line of the case contains x' and y', where (x', y') indicates the coordinates of the lamp.

To make the problem simple, you may assume that the input meets the following conditions:

- All coordinate values are integers not greater than 100 in their absolute values.
- No two walls intersect or touch except for their ends.
- The walls do not intersect nor touch each other.
- The walls turn each corner by a right angle.
- The lamp exists strictly inside the room off the wall.
- The x-coordinate of the lamp does not coincide with that of any wall; neither does the y-coordinate.

The input is terminated by a line containing a single zero.

Output

For each case, output the length of the unilluminated part in one line. The output value may have an arbitrary number of decimal digits, but may not contain an error greater than 10^{-3}.

Sample Input

```
4
0 0
2 0
2 2
0 2
1 1
6
2 2
2 5
0 5
0 0
5 0
5 2
1 4
0
```

Output for the Sample Input

```
0.000
3.000
```