Carnegie Mellon University

Invitational Programming Competition

Ten Problems

March 25, 2006

You can program in C, C++, or Java; note that the judges will re-compile your programs before testing.

Your programs should read the test data from the standard input and write results to the standard output; you should not use files for input or output.

All communications with the judges should be through the PC^2 environment.

Problem A: Numeric auction

The Country of Mobile Units (CMU) is ahead of other countries in the use of mobile communications. Its people not only carry cell phones, but also embed cellular devices into home appliances, and use their phones as universal remotes. For example, a person living in CMU can call her thermostat to turn on heating, or call her television set to change channels. When a CMU family switches to a new phone provider, it has to change phone numbers of all its appliances, which is very inconvenient. To eliminate this problem, the government has declared that every citizen can buy phone numbers and then use them with any provider. The initial sale is through an auction, which allows citizens to bid on their favorite numbers. The rules for submitting bids are as follows:

- A bid includes a specific phone number and a payment offered for it; for example, a bidder may offer $100 for the number 1234567.
- A phone number may include from one to hundred digits; thus, different numbers may have different length.
- Every citizen may submit any number of bids; if some of these bids win, he or she must pay the total of the winning bids.

After the CMU citizens submit all their bids, the government selects the winning bids, which must satisfy two constraints:

- If several bids are for the same phone number, at most one of them can win. This rule ensures that no number will belong to multiple people.
- The phone number in a winning bid cannot be a beginning of the number in another winning bid; for example, if some bid is for 012, and another is for 01234, then at most one of them can win. This rule ensures that no phone number will be a prefix of another, thus preventing wrong phone connections due to partially dialed numbers.

Given these constraints, the government needs to choose winnings bids with the maximal total payment. Your task is to write a program that helps with this task; that is, it finds a selection of winners that maximizes the total payment.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a list of bids, where each bid is on a separate line. A bid consists of a nonempty string and an integer, with a single space between them, and no spaces before the string or after the integer. The string represents a phone number; it includes at most 100 characters, where each character is a decimal digit. Note that it may have leading zeros, which cannot be ignored; for example, 012 and 12 are different phone numbers. The integer after the string is a payment offered for this phone number, which is between 1 and 100,000. The total number of bids in a test case is between 2 and 2,006; the last line of a test case is "0 0", which does not represent a bid. The last line of the input, after the last test case, is "−1 −1".

Output

For each test case, the output is a single integer on a separate line, which represents the total amount of the winning bids for the selection that maximizes this amount.

Sample input

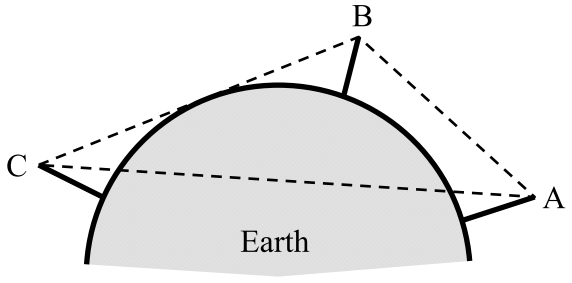
```
45 2
239 4
0 0
45 1
45 2
239 2
239 4
0 0
2 4
23 8
238 4
239 4
0 0
2 8
23 2
24 6
24 2
238 2
239 2
245 2
246 2
0 0
–1 –1
```

Sample output

```
6
6
8
10
```

Problem B: Monkeys and unicorns

The Country of Monkeys and Unicorns (CMU) is a sparse forest. Each monkey lives on its own tree, which serves not only as home, but also as a means of communication. When two monkeys need to talk, they climb to the tops of their trees and use gestures to communicate. Since the forest is sparse, the vegetation does not block the view, and two monkeys can see each other as long as the line between their tree tops does not cross the Earth surface. For example, monkeys *A* and *B* in the picture can talk, whereas monkeys *A* and *C* cannot. If the line between the tree tops is a tangent to the Earth surface, the monkeys do *not* see each other; for example, monkeys *B* and *C* cannot talk. Your task is to write a program that determines whether the monkeys living on two given trees can talk. You should assume that the surface of Earth is a perfect sphere with radius 20,900,000 feet. As a side note, the unicorns are irrelevant to this problem, but it is nice to have them in the forest.



Input

The input includes multiple test cases, each on a separate line; the number of cases is at most 2,006. Each test case includes three positive integers on the same line, separated by single spaces, with no spaces before the first integer or after the third. The first two integers are heights of two trees in feet, which are between 1 and 1,000; the third is the distance between these trees in feet, which is between 1 and 1,000,000. We measure this distance along the Earth surface between the lower parts of the trees rather than along the straight line. The last line of the input is "—1 —1 —1", which is not a test case.

Output

For each test case, the output is either "*visible*" or "*too far*", which should be on a separate line. If the monkeys living on the two given trees can see each other, the output is "*visible*"; else, it is "*too far*".

Sample input

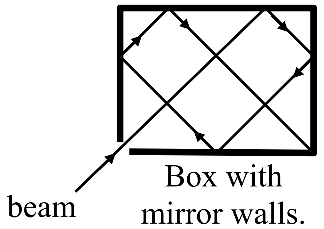
```
1 2 15608
2 1 15609
239 932 297323
932 239 297324
—1 —1 —1
```

Sample output

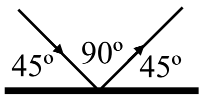
```
visible
too far
visible
too far
```

Problem C: Measuring C

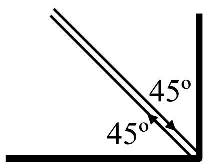
When freshmen take a relativity-theory course, they sometimes do not quite believe that the speed of light c is really 186,282 miles per second. To convince them, the instructor has built a C-Measuring Unit (CMU), which enables students to measure the speed of light. It consists of a light source, high-precision timer, and empty rectangular box with mirror walls, which has a small hole in the lower left corner (see the picture on the left). The light source sends a very short beam into the hole at 45° angle. When the beam hits a wall, it turns 90° degrees, as shown in the middle picture; when it hits a corner, it turns 180°, as shown in rightmost picture. The beam thus reflects many times, and eventually exits the box through the same hole in the lower left corner; the timer measures how long the beam was inside the box. If students know the distance traveled by the beam inside the box and the duration of its travel, they can calculate the speed of light. The missing part of this equation is the distance, and your task it to write a program that determines it.



Box with mirror walls.



Reflection from a wall.



Reflection from a corner.

Input

The input includes multiple test cases, each on a separate line; the number of cases is at most 2,006. Each test case includes two integers, *m* and *n*, which are between 1 and 1,000. These integers are on the same line, separated by a single space, with no spaces before the first integer or after the second. The two integers represent the box size; specifically, the box is $\sqrt{2} \cdot m$ millimeters long and $\sqrt{2} \cdot n$ millimeters wide. The last line of the input is "–1 –1", which does not represent a test case.

Output

For each test case, the output is either an integer or the words "*too large*", on a separate line. If the beam travels between 1 and 1,000,000 millimeters before leaving the box, the output is the traveled distance; else, it is "*too large*". Note that the traveled distance is always an integer, so the output should not involve rounding.

Sample input

```
1 1
300 200
239 932
999 1000
–1 –1
```

Sample output

```
4
2400
890992
too large
```

Problem D: In search of perfection

A *perfect number* is a positive integer that is exactly half of the sum of its divisors. For example, 6 is a perfect number; its divisors are 1, 2, 3, and 6, which satisfy the definition of perfection: 6 = (1 + 2 + 3 + 6) / 2. As another example, 28 is also a perfect number; its divisors are 1, 2, 4, 7, 14, and 28. Your task is to write a program that determines whether a given number is perfect.

Input

The input is a list of distinct positive integers between 1 and 1,000,000,000, each on a separate line with no surrounding spaces; the total number of integers is at most 2,006. The last line is "–1", which does not represent an input integer.

Output

The output is a list of perfect numbers contained in the input, in the same order, one number per line.

Sample input

```
6
2
39
28
239
496
–1
```

Sample output

```
6
28
496
```

Problem E: Treasure hunt

When pirates find buried treasures on a deserted island, they fight mercilessly for these treasures…or so most people believe. Although pirates spread rumors about their fighting, they are actually reasonable people, who try to avoid violence. When pirates need to divide treasures, they use the game called the Center of Mass Usurpation (CMU), which satisfies strict safety regulations of the Pirate Union.

To play this game, the pirates place the treasures in multiple known locations on the island, and then each pirate selects his or her initial location, which may be different from the locations of treasures. Note that several treasures may be in the same location, and several pirates may also be in the same location.

At each step of the game, the pirates determine the current ownership of the treasures. A pirate owns a specific treasure if he or she is the closest to it, and no other pirate is equally close to the same treasure. Note that, if several pirates are equally close to the same treasure, and no pirate is strictly closer, then the treasure does not belong to anyone.

After determining the ownership, each pirate moves to the center of mass of his or her treasures; that is, if a pirate owns treasures at locations (x_1, y_1), (x_2, y_2),…, (x_n, y_n), then he or she moves to the location $(\frac{x_1 + x_2 + \cdots + x_n}{n}, \frac{y_1 + y_2 + \cdots + y_n}{n})$. If a pirate does not own any treasures, he or she does not move.

After moving, the pirates determine the new ownership and then move again; they repeat this step until either getting tired or reaching a stable configuration, which does not lead to further movement. If the movement stops before pirates get tired, then they share the treasures according to the final ownership, and then share nobody’s treasures equally. If they get tired, then they declare a draw and share all treasures equally.

Your task it to write a program that determines the number of steps until the end of a CMU game based on the initial locations of treasures and pirates.

Input

The input includes multiple test cases; the number of cases is at most 20. A test case begins with a line that contains a single integer between 2 and 100, which represents the number of steps until pirates get tired. The next several lines are coordinates of treasures; each line includes two positive integers, separated by a single space, which represent x and y coordinates of a treasure. The last line in the treasure list is “0 0”, which does not represent a treasure. The lines after the treasure list are initial locations of pirates, in the same format as the treasure locations. The last line in the pirate list is also “0 0”, which does not represent a pirate. The last line of the input, after the last test case, is “−1”. The number of treasures in each test case is between 2 and 20, the number of pirates is between 2 and 5, and all input coordinates are integers between 1 and 1,000; however, note that the new coordinates of pirates during the game may not be integers.

Output

For each test case, the output is a single integer on a separate line, which represents the number of steps until the pirates either get tired or reach a stable configuration. For example, if the initial locations of pirates form a stable configuration, then the number of steps in the game is 0. As another example, if the first two steps lead to movement, but the third does not, then the number of steps is 2.

Sample input

```
10
1 1
2 2
0 0
1 1
2 2
0 0
10
1 1
1 2
1 3
1 4
1 5
1 6
1 7
0 0
1 1
3 1
0 0
−1
```

Sample output

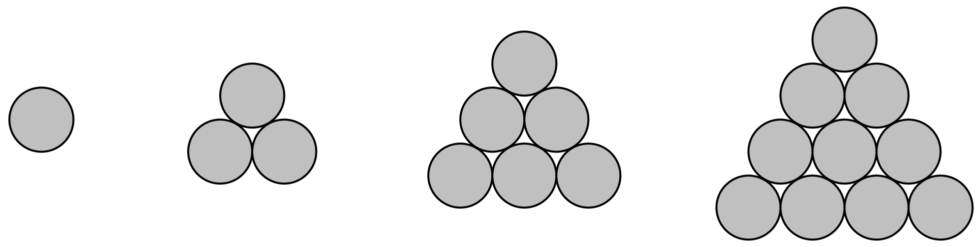
```
0
4
```

Problem F: Mystery of the Great Pyramid

The Great Pyramid of Egypt is probably one of the greatest historic wonders. Its construction technology was so far ahead of its time that some scientists have speculated about involvement of extra-terrestrials. Although aliens were indeed involved in building the pyramid, in reality they did not provide any technological help.

The story of the Great Pyramid began when aliens from the Constellation of Medium Ursa (CMU) landed in Egypt and taught Khufu the pharaoh to play baseball. Khufu liked the game so much that he decided to build a huge pyramid of balls to honor his CMU friends. According to his plan, the top layer of the pyramid contained one ball, the next layer was a triangle consisting of three balls, the third layer from the top was a six-ball triangle, and so on, as shown in the picture; in general, the *n*th layer from the top was an equilateral triangle with *n*-ball sides. Unfortunately, the required number of balls turned out prohibitive, and the pharaoh had to replace the original design with a square stone pyramid, which is now known as the Great Pyramid. Your task is to write a program that determines the number of balls in the original design.

As a side note, the Constellation of Medium Ursa does not actually exist. Since extra-terrestrials do not allow disclosure of their true location, we cannot tell you whether they came from Ursa Major or Ursa Minor.


Top layer. Second layer. Third layer. Forth layer.

Input

The input includes multiple test cases; the number of cases is at most 2,006. Each test case is an integer between 1 and 2,006, on a separate line with no surrounding spaces, which represents the number of layers in a pyramid. The last line of the input is “−1”, which does not represent a test case.

Output

For each test case, the output is a single integer on a separate line, which represents the number of balls in a pyramid with the given number of layers.

Sample input

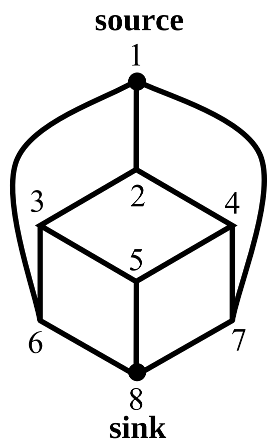
```
2
3
9
239
–1
```

Sample output

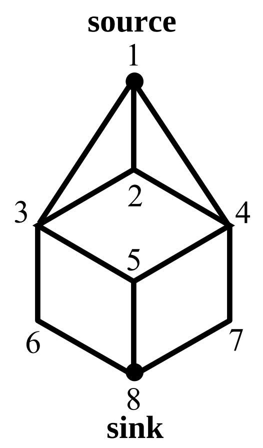
```
4
10
165
2303960
```

Problem G: Odd waterworks

The mayor of the City of Magic Unknown (CMU) likes odd waterworks, but he does not like odd numbers; he often orders construction of complex systems of fountains, and he always checks that the number of fountains in every composition is even. His latest project is a mesh of transparent pipes that carries water downhill; it consists of pipe segments and connectors that join them (see the picture). The water flows into this mesh through a single "source" connector at the hilltop, and it is collected into a single "sink" connector at the bottom, but it is split into many parallel pipes on the hillside. If you are watching a bubble moving down the pipes, you cannot predict its path, since you do not know which way it will turn at each connector. Since the mayor does not like odd numbers, he wants to ensure that every possible path from the source to the sink includes an even number of pipes. For example, the waterworks on the left of the picture satisfy this requirement, whereas the waterworks on the right have several odd-length paths, such as 1-3-6-8. Your task is to write a program that checks whether a given mesh of pipes has an odd path.



Waterworks with no odd-length paths from the source to the sink.



Waterworks that have odd-length paths.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a draft of waterworks, which describes pipes and connectors. The first line of a test case contains a positive integer *n* between 2 and 100,000, with no surrounding spaces, which represents the number of connectors. The connectors are numbered from 1 to *n*, where 1 is the topmost connector, which serves as the source of all water, and *n* is the sink of all water at the bottom of the hill. The following lines represent pipes, one pipe per line; the number of pipes is between 1 and 100,000. The description of a pipe includes two integers, separated by a single space, where the first integer is the connector at the higher end of the pipe, and the second is the connector at the lower end. For example, the line "2 4" represents a pipe that carries water from connector 2 to connector 4. The last line of a test case is "0 0", which does not represent a connector. All test cases describe valid waterworks, which means that every connector except the source has at least one incoming pipe, and every connector except the sink has at least one outgoing pipe; note that valid waterworks always have at least one path from the source to the sink. The last line of the input, after the last test case, is "–1".

Output

For each test case, the output is either "*valid*" or "*odd*", which should be on a separate line. If all paths from the source to the sink include an even number of paths, the output is "*valid*"; else, it is "*odd*".

Sample input

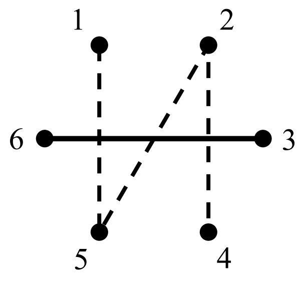
```
8
2 3
2 4
3 5
3 6
4 5
4 7
1 6
1 2
1 7
5 8
6 8
7 8
0 0
8
2 3
2 4
3 5
3 6
4 5
4 7
1 3
1 2
1 4
5 8
6 8
7 8
0 0
–1
```

Sample output

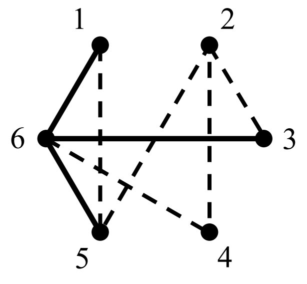
```
valid
odd
```

Problem H: Hexagon Game

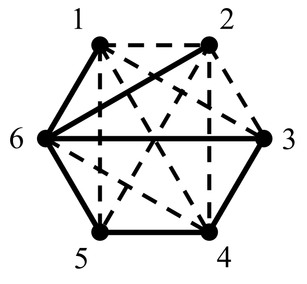
The *hexagon game* involves two players who gradually construct a six-vertex undirected graph with solid and dashed edges; Player 1 adds solid edges, whereas Player 2 uses dashes. The players begin with some initial six-vertex graph, which already has a few solid and dashed edges (see the picture on the left), and add new edges one by one; Player 1 makes the first move. At each move, a player has to add a new edge between two vertices that are not connected by any old edge. If Player 1 constructs a solid-line triangle, he loses the game; similarly, a dashed triangle means a loss of Player 2. For example, if the game ends as shown in the rightmost picture, then Player 2 has lost since he has constructed the triangle "1-2-5." Note that this game cannot end in a draw because a full six-vertex graph always has a solid or dashed triangle. Your task is to write a program that determines which player wins the game if both players use a perfect strategy.



Example of an initial position.



Example of a mid-game position; Player 1 makes the next move.



Example of a final position; Player 1 has won.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a list of edges in the initial position, which may include between 1 and 14 edges. The description of an edge consists of two distinct integers between 1 and 6, followed by the word "*solid*" or "*dashed*". Each edge is on a separate line, with one space between the two integers and one space between the second integer and the word. The integers represent the endpoints of the edge, and the word represents the edge type. The graph does *not* include multiple edges connecting the same pair of vertices, and the edges do *not* form any solid or dashed triangles. The last line in the graph description is "0 0 *end*", which does not represent an edge. The last line of the input, after the last test case, is "−1 −1 *end*".

Output

For each test case, the output is either "1" or "2" on a separate line. If Player 1 wins, the output is 1; else, it is 2.

Sample input

```
1 2 solid
0 0 end
2 1 dashed
0 0 end
6 1 solid
6 2 solid
6 3 solid
6 4 solid
6 5 solid
0 0 end
–1 –1 end
```

Sample output

```
2
1
2
```

Problem I: Precious time

The Clock Maker Union (CMU) is a group of craftspeople who make outrageously expensive timepieces. Their most popular product is a watch with two diamonds, embedded at the top and bottom of the dial. The CMU members keep an inventory of diamonds used in making these watches, and they maintain a database of available diamonds; for each diamond, the database includes its weight in milligrams an diameter in microns. When a CMU member makes a new watch with diamonds, he or she selects two most similar diamonds from the inventory. The similarity of diamonds is determined by the difference in their weight, with ties broken by the difference in diameter. Thus, the CMU member chooses the diamonds with the smallest difference of weights; if there are several pairs of diamonds with the same smallest difference, he or she chooses the pair with the smallest difference in diameter among them. You need to write a program that helps with this task; that is, it identifies two most similar diamonds in a database.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a list of diamonds, which may include between 2 and 100,000 elements. The description of a diamond consists of two integers between 1 and 100,000, on a separate line, with a single space between them. The first integer is the weight of the diamond, and the second is its diameter. The last line in a test case is "0 0", which does not represent a diamond. The last line of the input, after the last test case, is "—1 —1".

Output

For each test case, the output includes two natural numbers on a separate line, with a single space between them. The first number is the weight difference between two most similar diamonds, and the second is the diameter difference between them.

Sample input

```
1 8
2 7
4 6
6 6
0 0
1 5
2 4
3 2
4 3
0 0
1 2
1 3
2 1
2 2
0 0
1 1
1 1
0 0
–1 –1
```

Sample output

```
1  1
1  1
0  1
0  0
```

Problem J: Magic of time travel

Fred Weasley and George Weasley are two young magicians from the “Harry Potter” book series. In the fifth book, they have opened a shop, where they invent and sell magic novelty items. Their latest invention is a time machine, which allows jumping several years forward or backward. Since their customers are magicians, who do not have experience with mechanical or electronic interfaces, Fred and George have developed a very simple interface. Specifically, a time machine has two buttons; the first button allows moving *n* years to the future, and the other is for moving *n* years to the past. For each machine, *n* is a hard-wired constant, called the “jump number” of the machine. The user can press the buttons multiple times, thus traveling *n*, 2*n*, 3*n*, 4*n*, etc. years.

To promote this product, Fred and George have given several machines with different jump numbers to Harry Potter and his friends. Harry’s friends want to try going into a very distant future; however, Hermione Granger, who is the only one among them with reasonable knowledge of nonmagic sciences, has pointed out two restrictions on their travel. First, since the Sun is gradually expanding, the Earth is likely to become uncomfortably hot in about 1.1 billion years, which means that they should not go further than that. Second, they should select a year where they all can meet, which means that it should be reachable by all available machines. Given these restrictions, they want to go as far into the future as possible. Your task is to write a program that determines how many years into the future they can travel.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a list of distinct integers between 1 and 2,006, which represent the jump numbers of available machines; each integer is on a separate line, with no surrounding spaces. The number of time machines in a test case is between 2 and 2,006; the last line in a test case is “0”, which does not represent a machine. The last line of the input, after the last test case, is “−1”.

Output

For each test case, the output is an integer on a separate line, which shows how far into the future Harry and his friends can travel. This integer is the largest value between 0 and 1,100,000,000 that is divisible by the jump numbers of all machines. Note that, if there is no positive integer that satisfies these constraints, then the output is 0, which means that Harry and his friends should not time-travel.

Sample input

```
1
4
5
0
2
3
9
0
239
932
0
2001
2003
2005
0
–1
```

Sample output

```
1100000000
1099999998
1099929624
0
```