Carnegie Mellon University

Invitational Programming Competition

Eight Problems

March 31, 2007

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: Flight distances

Airline companies keep track of the distance traveled by each plane in order to ensure timely maintenance. Your task is to write a program that helps to determine these distances; specifically, it should input the coordinates of the airports visited by a specific plane and compute the overall flight distance. For simplicity, assume that the earth surface is flat, and the plane always flies along a straight line between two airports.

Input

The input includes multiple test cases, which correspond to different planes; the number of cases is at most 20. Each test case is a list of airports, where each airport is on a separate line. An airport description consists of two positive integers between 1 and 10,000, with a single space between them, which are the airport coordinates in miles. The plane starts in the first of these airports, and visits the other airports in order, until reaching the last airport. The total number of airports in a test case is between 2 and 2,007; the last line of a test case is "0 0", which does not represent an airport. 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 number of miles traveled by the plane, rounded to the nearest integer.

Sample input

```
1 1
2 2
0 0
1 1
1 11
11 11
11 1
1 1
0 0
-1 -1
```

Sample output

```
1
40
```

Problem B: Feel the superpower

Alice and Bob are two mathematicians, who often collaborate in their number-theory research, but unfortunately they tend to disagree when selecting a scientific journal for publishing their results. To avoid related arguments, they play a game of chance, called "superpowers," and then the winner chooses the journal. The game is based on the following recursive definition of the *superpower* operation:

- For every positive integer n,
  $superpower(n, 1) = n$
- For every two positive integers n and m,
  $superpower(n, m + 1) = n^{superpower(n,\, m)}$

For example,

$$superpower(2, 3) = 2^{(2^2)} = 2^4 = 16$$

$$superpower(2, 4) = 2^{(2^{(2^2)})} = 2^{(2^4)} = 2^{16} = 65536$$

When Alice and Bob play the game of superpowers, they randomly select two natural numbers m and n, and then compare $superpower(m, n)$ with $superpower(n, m)$. If the first value is greater than the second, Alice wins; if it is smaller, Bob wins; finally, if the two values are equal, it is a draw. Unfortunately, Alice and Bob do not have much experience with programming, and they have to compare these values manually. Your task is to help them by writing a comparison program.

Input

The input includes multiple test cases; each case is on a separate line, and the number of cases is at most 20. A test case includes two integers between 1 and 5, with a single space between them, which are the values of m and n. The last line of the input is "−1 −1", which does not represent a test case.

Output

For each test case, the output is the word "*smaller*", "*greater*", or "*equal*" on a separate line, which represents the result of comparing $superpower(m, n)$ with $superpower(n, m)$.

Sample input

```
1 2
2 2
2 4
4 5
-1 -1
```

Sample output

```
smaller
equal
greater
greater
```

Problem C: Xxigle from Xxigle

The Xxigle civilization is an advanced alien culture, which has evolved on the Planet of Xxigle, in the warm sunshine of the Xxigle Sun. Xxiglian people look very similar to Earth people, which allows them to visit Earth and pass for humans during their visits. When young Xxigles stay on Earth, they sometimes enroll in Earth universities, and sometimes even earn local degrees. Since the intellectual abilities of Xxiglian people are far superior to the human abilities, they can easily take a lot of hard courses and get excellent grades. In fact, if you have a friend who always gets straight As, he or she is probably from Xxigle.

Most Xxiglian students can take an unlimited number of Earth courses per semester, and the only obstacle that prevents them from graduating in one semester is the university rules regarding course prerequisites. For example, if course 101 is a prerequisite of 201, course 201 is a prerequisite of 301, and course 301 is part of the degree requirements, then even the smartest Xxigle needs at least three semesters to graduate. As another example, if course 101 is a prerequisite of 201, and course 201 is a prerequisite of 101, then students cannot graduate at all.

Your task is to write a program that analyzes prerequisite rules and determines the minimal number of semesters required for earning a degree. Assume that a Xxigle student can take any number of courses per semester, but she should follow all prerequisite rules, and she cannot take a course and its prerequisite in the same semester. For simplicity, also assume that she has to take *all* available courses.

Input

The input includes multiple test cases, which correspond to different university programs; the number of cases is at most 20. Each test case is a list of prerequisite rules, where each rule is on a separate line. A rule description consists of two positive integers between 101 and 999, with a single space between them, which represent course numbers; the first course in a rule is a prerequisite of the second course. The total number of rules in a test case is between 1 and 2,007; the last line of a test case is “0 0”, which does not represent a rule. The last line of the input, after the last test case, is “−1 −1”.

Output

For each test case, the output is either an integer or the words “*inconsistent rules*”, on a separate line. If the prerequisite rules allow students to graduate, then the output is the smallest number of semesters required for graduation. If the prerequisite rules do not allow graduation, then the output is “*inconsistent rules*”.

Sample input

```
101 201
202 301
0 0
101 201
301 401
0 0
101 201
101 301
201 401
301 401
401 501
0 0
101 201
201 101
0 0
−1 −1
```

Sample output

```
3
2
4
inconsistent rules
```

Problem D: Prime time

A *prime number* is an integer number, strictly greater than 1, that is divisible only by 1 and itself; for example, 23 and 239 are prime numbers, whereas 0, 1, 4, and 2,007 are not primes. The first prime number is 2, the second is 3, the third is 5, and so on. Your task is to write a program that finds the *n*th prime number for a given *n*.

Input

The input is a list of distinct positive integers between 1 and 1,000,000, each on a separate line with no surrounding spaces; the total number of input integers is at most 2,007. The last line is "–1", which does not represent an input integer.

Output

For each input value *n*, the output is the *n*th prime number, on a separate line.

Sample input

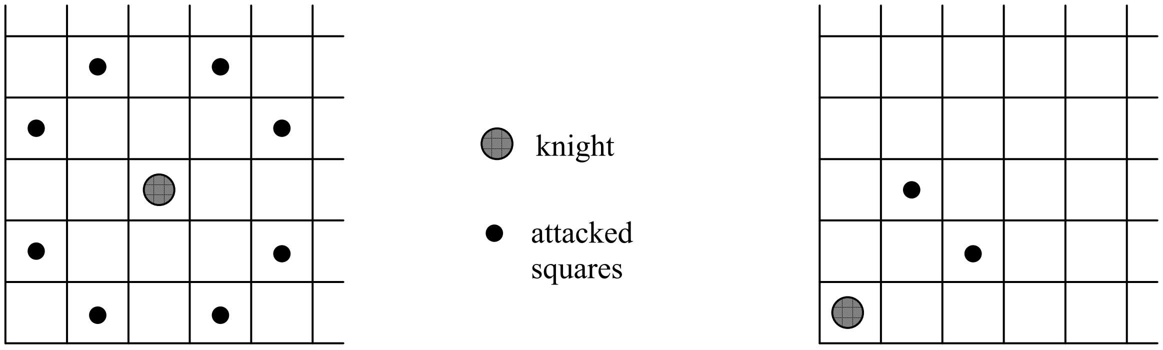
```
2
3
23
239
10000
–1
```

Sample output

```
3
5
83
1499
104729
```

Problem E: Peaceful knights

A knight in the chess game is a piece that attacks eight squares around it as shown in the picture on the left. If some of these squares are outside the chessboard, then it attacks fewer squares; for instances, the knight in the picture on the right attacks only two squares. Your task is to determine the maximal number of knights that can be placed on the square board of a given size, in such a way that no knight attacks any other.



Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is an integer between 1 and 1,000, on a separate line with no surrounding spaces, which represents the size of a square chessboard; for example, if this integer is 8, then the board is 8 × 8. 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 maximal number of knights that can be placed on the board without attacking each other.

Sample input

```
1
2
3
4
-1
```

Sample output

```
1
4
5
8
```

Problem F: Round table

King Arthur once invited a number of knights to his castle, where they stayed for several days. Each evening, the king and his guests dined at the Round Table. According to the king's decree, they took different seats on different evenings, and *no two people sat next to each other more than once*. When the knights could no longer satisfy this decree, they left the king's castle. What is the maximal number of days they could have stayed in the castle? Your task is to write a program that answers this question.

Input

The input includes multiple test cases; the number of cases is at most 20. Each test case is a *prime number* between 2 and 1000, on a separate line with no surrounding spaces, which represents the number of people at the table. Note that this number is *prime* in all test cases. 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 maximal number of seating arrangements that satisfy the king's decree.

Sample input

```
2
3
5
239
–1
```

Sample output

```
1
1
2
119
```

Problem G: Dog racing

Jim is a forest ranger, and he always takes his dog with him to work. Every morning, Jim and his dog go to the office to complete the daily paperwork, and then spend the rest of the day in the forest. Jim lives in a few miles from the office, and he walks to the office through the woods, while his dog impatiently runs back and forth.

When Jim leaves his home, the dog runs ahead of him all the way to the office and then back from the office to Jim. After finding Jim, the dog again runs to the office, and then again back to Jim, and so on; it continues running to the office and back until Jim reaches the office. Clearly, the distance covered by the dog is greater than the distance from Jim's home to the office. Your task is to write a program that computes this distance.

Input

The input includes multiple test cases, each on a separate line; the number of cases is at most 20. A test case includes three integers between 1 and 100, separated by single spaces; the first integer is the distance from Jim's home to his office in miles, the second is Jim's speed in miles per hour, and the third is the speed of the dog, also in miles per hour, which is strictly greater than Jim's speed. The last line of the input is "−1 −1 −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 miles covered by the dog, rounded to the nearest integer.

Sample input

```
2 3 5
1 99 100
-1 -1 -1
```

Sample output

```
3
1
```

Problem H: Heffalump hunt

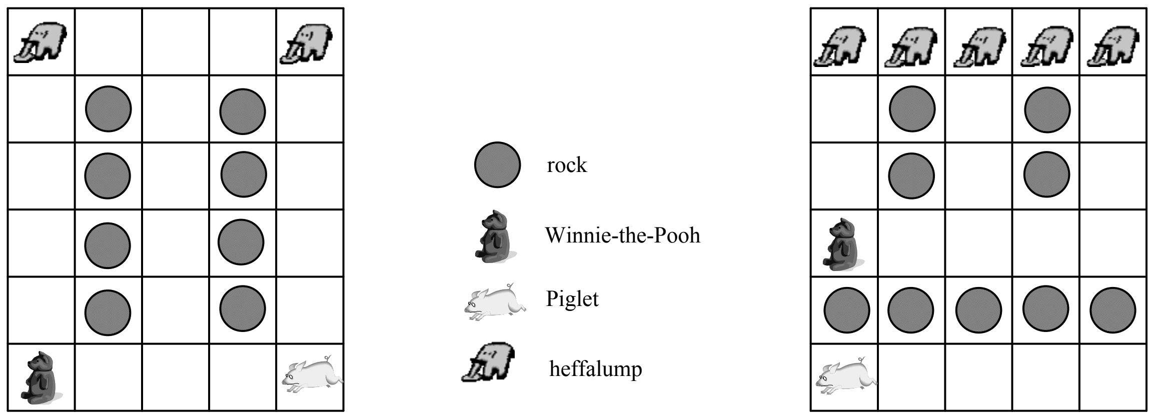
If you have read about Winnie-the-Pooh, you may remember that Pooh and his friend Piglet are trying to catch a heffalump, which is a large animal that may or may not look like an elephant. Initially, they tried to use a Cunning Trap, which was a Very Deep Pit with honey jars at the bottom, but this approach did not work; apparently, heffalumps are sufficiently smart to avoid this trap.

Eventually, Pooh and Piglet have decided to go into Heffalump Valley, which is a maze of large rocks, and try to corner a heffalump. Since heffalumps run twice faster than bears and pigs, Pooh and Piglet should carefully plan their strategy. Your task is to write a program that determines whether they will succeed.

We represent the valley by a rectangular greed, where some squares are empty, and some contain rocks; we give two examples of a valley in the picture. We know the initial positions of Pooh, Piglet, and all heffalumps that live in the valley, and we assume that neither the two hunters nor the heffalumps leave the valley during the hunt.

Pooh, Piglet, and heffalumps can move only among empty squares, and they cannot climb rocks. At each move, Pooh, Piglet, and heffalumps can remain in their current squares or move to one of the four adjacent squares, as long as these squares are within the valley and do not contain rocks. The hunt is a sequence of steps, where each step includes one move by Pooh, one move by Piglet, and two moves by each heffalump. First, Pooh makes his move, then Piglet makes his move, and then each heffalump makes its two moves; note that Pooh, Piglet, and several heffalumps may be on the same square at the same time.

Pooh and Piglet are trying to catch a heffalump, whereas all heffalumps are trying to evade them. We assume that the hunters and heffalumps always know the positions of each other. If either Pooh or Piglet ever shares a square with a heffalump, then the two friends have caught this heffalump and succeeded in their hunt. Your program should determine whether they can eventually catch a heffalump or all heffalumps can indefinitely avoid the capture.



Input

The input includes multiple test cases, which correspond to different layouts of the valley; the number of cases is at most 20. The first line of a test case includes three integers, *n*, *m*, and *k*, which are between 2 and 10; the first two integers represent the length and width of the valley, and the third is the number of heffalumps. The rest of the test case consists of *m* lines, each containing exactly *n* characters, which represent the layout of the valley. We denote empty squares by "–", rocks by "*x*", the position of Winnie-the-Pooh by "*w*", the position of Piglet by "*p*", and the position of each heffalump by "*h*". Initially, the two hunters and all heffalumps are on different squares; thus, the map includes one letter "*w*", one letter "*p*", and *k* letters "*h*". The last line of a test case is blank, which is not part of the map. The last line of the input, after the last test case, is "*end*".

Output

For each test case, the output is the word "*success*" or "*failure*" on a separate line, which represents the outcome of the hunt. If Pooh and Piglet can eventually catch a heffalump, the output is "*success*"; if all heffalumps can indefinitely evade them, it is "*failure*."

Sample input

```
5 6 2
h---h
-x-x-
-x-x-
-x-x-
-x-x-
w---p

5 6 5
hhhhh
-x-x-
-x-x-
w----
xxxxx
p----

end
```

Sample output

```
success
failure
```