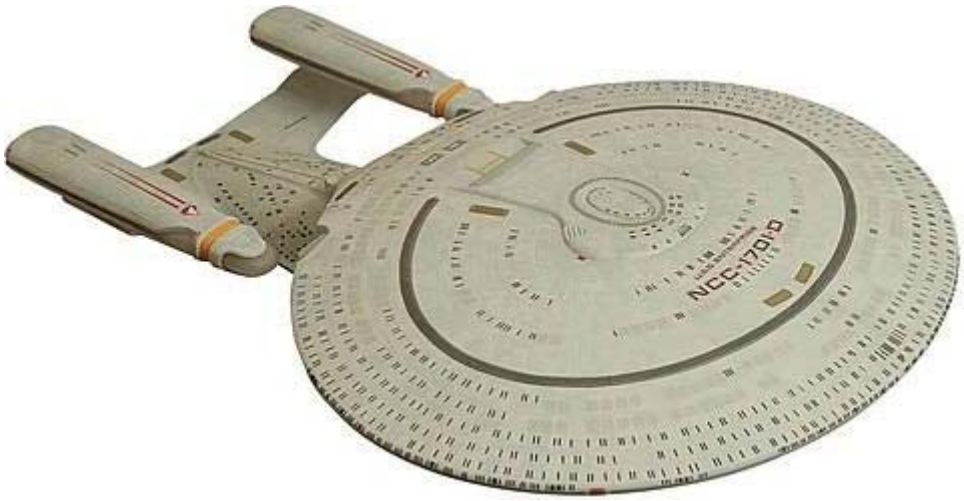
Carnegie Mellon University
Spring Programming Competition

Preliminary Version

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.

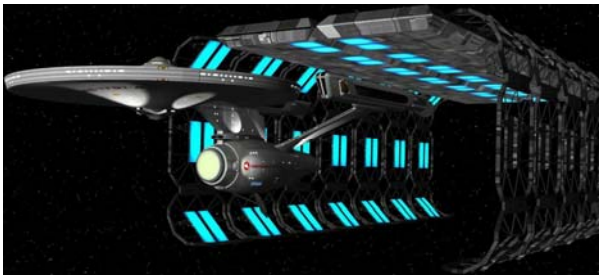
All communications with the judges should be through the PC^2 environment.



Welcome aboard the famous *USS Enterprise* (NCC-1701-D), a Galaxy Class starship that has been part of the Federation Starfleet since 2363.

While solving these problems, you will learn new facts about the *Enterprise* and its crew, which have not been revealed in the *Star Trek* television series.

Problem A: Takeoff



Ready for the takeoff?

Have you double-checked that the *Enterprise* is indeed ready?

When a starship is about to leave a Federation space dock and boldly go where no man, woman, or android has gone before, its captain must ensure that it has sufficient supply of all necessary equipment and materials, such as deuterium and anti-deuterium, dilithium crystals, trillium, photon and quantum torpedoes, and hand phasers. The list of standard supplies is very lengthy, but fortunately the captain does not have to go through it manually. Every Federation ship has an onboard computer, which can instantly determine whether the ship is ready for its mission. A human engineer has to write a program for checking supplies, and then the computer will execute it before every takeoff. Your task is to implement this program.

Input

The input includes multiple test cases, which describe different pre-flight situations; the number of cases is at most 20. A test case is a list of required supplies, one supply type per line; the number of lines in a test case is between 1 and 1000. Every line includes a lower-case alphabetical string, whose length is between 1 and 20, followed by two integer values between 1 and 1000, separated by single spaces. The string is the name of a supply type, which is unique within the test case; the first integer is the amount of this supply available onboard; and the second integer is the amount required for the mission. If the first number is strictly smaller than the second, the ship does not have enough of this supply. The last line of a test case is "`. 0 0`" (a period and two zeros, separated by single spaces), which does not represent a supply. The last line of the input, after the last test case, is "`end 0 0`".

Output

For each test case, the output is a single line. If all supplies are sufficient, the output is "`Ready for the takeoff`". If not, the output begins with the word "`Load:`", followed by the names of insufficient supplies, in the same order as in the input, separated by single spaces.

Sample input

```
deuterium 100 90
antideuterium 100 90
dilithium 80 80
trillium 21 20
photontorpedoes 200 200
quantumtorpedoes 500 1
phasers 1000 1000
. 0 0
poodles 999 1000
rottweilers 1 2
sehlats 1000 1000
targs 2 1
. 0 0
end 0 0
```

Sample output

```
Ready for the takeoff
Load: poodles rottweilers
```

Problem B: Beverley Crusher

Commander Beverley Crusher, M.D. (human) is the chief medical officer of the *Enterprise*. Since most of the *Enterprise* crew members are very healthy, Dr. Crusher usually sees only a few patients a day, and has plenty of time to work on academic research and participate in away missions. Occasionally, however, the *Enterprise* faces major medical emergences, and then Dr. Crusher has to work around the clock.

The latest emergency has begun after the crew returned from a shore leave on the planet of Risa. Dr. Crusher has discovered that most crew members caught the Risan flu, which is an annoying disease that makes a person sneezy, itchy, and sleepy for several days. Fortunately, Federation scientists have recently developed a medication for the Risan flu, which can prevent all its symptoms, but only if used on its early stage. The administering of this medication is a complex procedure, and Dr. Crusher is the only physician onboard who can do it properly. She needs five minutes per patient, and she may not have time to cure everybody.

Dr. Crusher has asked her assistants to evaluate the flu severity for every patient and determine how soon each patient needs to get the drug. It turns out that some need it right away, some can wait a few hours, and some with milder cases can wait even longer. Now Dr. Crusher has to triage her patients, that is, determine the order of attending to them that would enable her to cure the maximal number of people. Your task is to implement a program for the triage.

Input

The input includes multiple test cases, which describe different emergency scenarios; the number of cases is at most 20. A test case is a list of patients, each on a separate line; the number of patients in a test case is between 1 and 1000. The description of a patient is an integer between 1 and 1440, which is the number of minutes within which he or she needs to get the drug. For instance, if this number is 60, the patient needs to get it within an hour, which means that Dr. Crusher has to begin administering it within 55 minutes. The last line of a test case is "`0`" (zero), which does not represent a patient. 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 is the maximal number of patients that can be cured.

Sample input

```
10
10
10
10
0
5
10
15
20
0
4
9
14
19
0
1
1
1
1
0
5
0
−1
```

Sample output

```
2
4
3
0
1
```

Problem C: Wesley Crusher

Wesley Crusher (human) is the son of Dr. Beverly Crusher, and he travels with Dr. Crusher on the *Enterprise*. He is a talented and somewhat geeky teenager, who often entertains himself by building high-tech toys. His latest invention is an insect-sized robot for the cutting of rectangular plastic sheets, which crawls over a sheet and makes a cut along its path. The robot inputs a string of letters *L*, *R*, and *S*, which defines its path, and moves over a rectangular sheet according to the following rules (see example pictures on the next page):

- The robot starts in the lower left corner of the sheet, facing upward.
- At each step, the robot performs two actions:
  (1) It reads the next letter of the input, which determines whether it has to turn. If the letter is *L*, it turns left; if *R*, it turns right; if *S* (straight), it makes no turn.
  (2) It crawls one inch in the resulting direction.
- After reading the last letter and completing its last step, the robot stops. Note that the overall length of its path (in inches) equals the length of the input string.

Wesley randomly generated a few strings and used them to test the robot, but realized that random strings usually give uninteresting results. The robot sometimes stops in the middle of the sheet without cutting it in two, sometimes crawls repeatedly over the same spot, and sometimes falls off the sheet. After getting these disappointing outcomes, Wesley has concluded that a string produces an interesting result only if it satisfies the following constraints:

- At the end, the robot returns to its starting point; that is, it stops in the lower left corner of the sheet.
- The robot never visits the same point twice, with the exception of returning to its starting point.
- The robot never leaves the sheet; however, it can crawl along the sheet's edges.

Your task is to help Wesley select "interesting" inputs. Specifically, you should implement a program that reads a potential input string and checks whether it satisfies Wesley's constraints.

Input

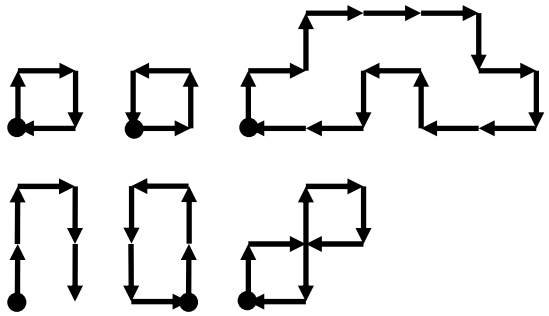
The input includes multiple test cases, which represent different plastic sheets and input strings; the number of cases is at most 1000. A test case comprises two integers and a string of capital letters *L*, *R*, and *S*, all on the same line. It includes a single space between the two integers, and another single space between the second integer and the string. The first integer is the width of the sheet (in inches), that is, its size along the horizontal dimension, and the second is the sheet height; both integers are between 1 and 1000. The string represents the robot's input; its length is between 4 and 4000. The last line of the input, after the last test case, is "`0 0 .`" (two zeros and a period).

Output

For each test case, the output is a single line. If the robot's path satisfies all three constraints, the output is "`interesting`"; else, it is "`uninteresting`".

Sample input

```
1 1 SRRR
10 10 RLLL
1000 1000 SRLRSSRLRRSRLLRS
2 2 SSRRS
2 2 SSLLSL
2 2 SRLRRRLR
0 0 .
```



Sample output

```
interesting
interesting
interesting
uninteresting
uninteresting
uninteresting
```

Problem D: Data

Lieutenant Commander Data (android) is the only robotic member of the *Enterprise* crew. While his positronic brain is inferior to the human mind in several important respects, such as the ability to experience emotions, it is superior on most technical problems. Data thinks with the speed of a computer and near-instantly completes all intellectual tasks involved in his duties, which means that he is bored most of the time. To alleviate the boredom, he occasionally invents puzzles to occupy his mind. For instance, Data likes to play the *find-squares* game, which involves identifying squares in images of a star sky. He would briefly look out of the window and memorize the view of the sky, and then he would count all squares formed by stars in the memorized image. He counts only nonzero-size squares, that is, he does *not* view a single star as a square. To verify the accuracy of his counting, he needs a computer program that would solve the same problem and tell him the correct answer. Your task is to implement this program.

Input

The input includes multiple test cases, which correspond to different sky images; the number of cases is at most 20. A test case is a list of stars, each on a separate line; the description of a star consists of two integers between 1 and 1000, separated by a single space, which represent its Cartesian coordinates in the image. The number of stars in a test case is between 4 and 4000, and they all have distinct coordinates. The last line of a test case is "`0 0`" (two zeros, separated by a single space), which does not represent a star. The last line of the input, after the last test case, is "`−1 −1`".

Output

For each test case, the output is an integer on a separate line, which is the number of distinct squares formed by the stars.

Sample input

```
1 1
1 6
1 11
6 1
6 6
6 11
11 1
11 6
11 11
0 0
1 2
2 4
3 1
4 3
0 0
1 1
2 2
3 3
4 4
0 0
−1 −1
```

Sample output

```
6
1
0
```

Problem E: Geordi La Forge

Lieutenant Commander Geordi La Forge (human) is the chief engineer of the *Enterprise*, and his main duty is the maintenance of the ship's warp drive. When diagnosing and fixing engine problems, he often has to crawl through *Jefferies tubes*, which are painfully narrow tunnels through the ship's critical systems. After spending over 2048 hours in those tubes and getting permanent sore spots on his knees and elbows, La Forge finally got around to building a robot that would do most of the crawling. This robot is a small flying ball with an anti-gravity engine (see the picture), powered by an onboard battery. It can float through the air along a Jefferies tube, transmit images and sounds to the human operator, and use its retractable arms for minor repairs.

Unfortunately, its engine quickly drains the battery and the robot cannot travel more than a few feet without recharging. To alleviate this problem, La Forge installed electric outlets in the tubes and enabled the robot to charge from them. He has later realized that he installed too many outlets and the robot may skip some of them. The related optimization task is to decide which outlets to use in order to minimize the number of recharging stops. To formalize this problem, La Forge has made the following assumptions:

- The robot has to travel the full length of a tube, from its entrance to its far end and then back to the entrance.
- When the robot enters the tube, its battery is fully charged.
- When the robot stops to recharge, it regains the full charge.
- The power level upon returning to the entrance does not matter; at that point, the robot may have no power or any partial charge.

Your task is to write a program that determines the minimal number of recharging stops.

Input

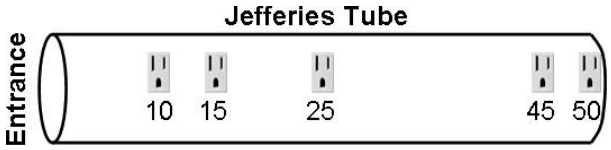
The input includes multiple test cases; the number of cases is at most 20. The first line of a test case is an integer between 1 and 100000, which is the maximal distance covered by a robot without recharging. The other lines specify electric outlets, one per line; the number of outlets in a test case is between 1 and 1000. The description of an outlet is a single integer between 1 and 100000, which is its distance from the tube entrance. These outlet distances are distinct and listed in increasing order. The last outlet is always at the very end of the tube, which means that its location determines the tube length. The last line of a test case is "`0`" (zero), which does not represent an outlet. The last line of the input after the last test case is "`-1`".

Output

For each test case, the output is a single line. If the robot can reach the far end of the tube and return to the entrance, the output is a single integer, which is the minimal number of recharging stops; else, the output is "`need more outlets`".

Sample input

```
100
10
15
25
45
50
0
60
10
15
25
45
50
0
20
10
15
25
45
50
0
10
10
15
25
45
50
0
−1
```



Sample output

```
0
1
6
need more outlets
```

Problem F: Jean-Luc Picard

Captain Jean-Luc Picard (human) is the captain of the *Enterprise*. If you have watched *Star Trek: The Next Generation*, you may have gotten the impression that his job mostly involves glorious and fun activities, such a blowing up enemy ships, leading dangerous away missions, and saving human and alien settlements from terrible disasters; however, nothing can be further from the truth. While the captain may occasionally do something heroic, his main job is the same as that of all high-ranking officials, which is paperwork.

Picard has to produce innumerable mission statements, supply requests, reports of encounters with alien races, crew evaluations, and so on. When he gets up in the morning, his first task is to decide which reports to write today and then determine the order of working on them. The Starfleet regulations specify which documents are to be completed before which others, thus creating numerous ordering constrains. For instance, he has to produce "justifications for the firing of individual photon torpedoes" before a "summary on the mission use of torpedoes", which in turn has to be done before a "request for additional torpedoes." As another example, he has to write an "abbreviated quarterly evaluation of the junior officers" before a "detailed quarterly evaluation of the senior officers", and only then he can fill out a "quarterly self-assessment report".

By the lunchtime, he is usually sick of his paperwork and desperately needs to take his mind off the Starfleet bureaucracy. If *Enterprise* happens to face a major crisis, such as a Romulan attack or a supernova explosion, then Picard uses this opportunity to put aside his reports and make a few brilliant split-second decisions. If there is no crisis, he looks for other distractions, such as helping Geordi fix engine problems, playing poker with Riker and Worf, or even helping Data count squares in sky images.

Today, Picard has invented a new unproductive activity for his lunch break. He has decided to count the number of all possible orderings of today's reports that would be consistent with the Starfleet regulations. After spending half-hour on that puzzle, he has realized that the number of orderings is very large, and there is no way to count them by hand. Picard has thought of asking Data's help, but Data has turned out to be unreachable. He is currently on an away mission, and the *Enterprise* has temporarily lost communications with that away team because of the super-anti-photonic-positronic hyper-warp fluctuations in the nearby space.

The only way to solve Picard's puzzle is to write a program that would count all possible orderings, but the captain does not have much experience with software engineering. Your task is to implement this program for him.

Input

The input includes multiple test cases; the number of cases is at most 20. A test case is a list of distinct ordering constraints, each on a separate line. A constraint consists of two lower-case alphabetical strings, separated by a single space, which represent specific documents; the length of each string is between 1 and 20. The first document in the constraint must be completed before the second; for example, the constraint "`juniorevaluation seniorevaluation`" indicates that the evaluation of the junior officers must be done before that of the senior officers. The input includes at least one constraint for each document, which means that the constraint list includes all documents. The number of constraints in a test case is between 1 and 1000. The last line of a test case is "`. .`" (two periods, separated by a single space), which does not represent a constraint. The last line of the input after the last test case is "`end .`".

Output

For each test case, the output is an integer on a separate line, which is the number of distinct orderings consistent with given constraints. You may assume that the number of orderings is at most 10000. Note that a constraint set may be inconsistent, and then the output must be "0".

Sample input

```
torpedojustification torpedosummary
torpedosummary torpedorequest
juniorevaluation seniorevaluation
seniorevaluation selfassessment
. .
a b
a c
a d
b c
b d
c d
. .
a b
b c
c d
d a
. .
end .
```

Sample output

```
20
1
0
```

Problem G: William Riker

Captain William T. Riker (human) is the second in command on the *Enterprise* and he occasionally gets the opportunity to be the acting captain while Picard is away. Riker likes these opportunities and tries to be an exemplary captain, but his lack of experience sometimes leads to blunders. His latest blunder happened when the ship was stationed near the planet of Risa (see the picture) and Picard took a short medical leave for a checkup of his artificial heart.

Riker had long noticed that the holodeck, which is the ship's virtual-reality facility, seemed to be the most dangerous place on the *Enterprise*. Its safety protocols frequently failed, trapping people inside the holodeck, causing injuries, and even creating artificial characters that posed a threat to the ship as a whole. While the *Enterprise* was waiting for its next mission on a Risan orbit, Riker decided to get to the bottom of the holodeck problems and assigned Geordi La Forge to check its software. La Forge readily found multiple bugs, legacy problems, viruses, worms, Trojan horses, spyware, and adware, some of which were going back several centuries to the days of Microsoft. He uncovered so many problems it was a wonder the holodeck had not yet killed anyone. To prevent further accidents, Riker shut down the holodeck and scheduled a full software reinstall during the next round of ship maintenance in a Federation dock.

Unfortunately, his decision led to a morale plunge, as most crew members relied on the holodeck as their main entertainment. To boost the morale, Riker let all nonessential personnel take a four-day shore leave on Risa. Since the *Enterprise* had a week until its next mission, Riker saw no harm in letting the crew enjoy themselves.

The next day, Picard unexpectedly returned from his leave and brought top-secret orders—so secret they could not be transmitted over any communication channel—that required the *Enterprise* to depart immediately to an Omega-quadrant through a newly discovered wormhole. Since most crew members were on the planet, Picard had to delay the departure and tasked the embarrassed Riker with getting everyone back.

As usual, the ship's transporter did not work, ostensibly because of solar flairs, but mostly because the episode authors wanted to show shots of a shuttlecraft flying over the planet. Riker hoped to gather all crew in one shuttle flight, but the task turned out far more complex. First, the number of crew members on the planet far exceeded the shuttle capacity. Second, a lot of them had caught various strains of Risan flu and had to be quarantined. The shuttle should not transport healthy and sick people together, and also it should not transport people with different flu strains together, to avoid spread of the infection. In one flight, it can carry only healthy people or only people with the same flu strain, up to its capacity. Given these constraints, Riker has to determine the minimal required number of flights. Your task is to implement a program for this task.

Input

The input includes multiple test cases, which describe different scenarios; the number of cases is at most 20. A test case is a list of integers between 1 and 1000, each on a separate line. The first integer is the shuttle capacity, the second is the number of healthy crew members on Risa, and the other values show the number of crew members with different flu strains. For instance, the first case in the sample input shows that the shuttle capacity is 10; the number of healthy people is 25; and the number of sick people with the first strain is 10, with the second strain is 5, and with the third strain is 51. The number of different strains is between 1 and 1000. The last line of a test case is "0" (zero), which does not represent a flu strain. 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 is the minimal required number of shuttle flights.

Sample input

```
10
25
10
5
51
0
239
1
1
0
1
239
239
0
2
1
2
3
4
5
6
7
8
0
−1
```

Sample output

```
11
2
478
20
```

Problem H: Deanna Troy

Lieutenant Commander Deanna Troy (half-Betazoid, half-human) is the ship counselor, whose duties are similar to those of a psychotherapist. She helps crew members address their social and psychological challenges, and she also works with senior officers to ensure that crew assignments do not lead to personal problems or conflicts of interest. In particular, her duties include screening the composition of away teams.

When the *Enterprise* explores a newly discovered planetary system, Captain Picard sends several away teams to each terrestrial planet. He assigns the leader of every team, and he also selects one of the team leaders on each planet to serve as the coordinator of all activities on that planet. After making a tentative team selection, the captain passes his roster to Troy, who checks that it does not create social problems. She uses her records of current and past marriages, current and past romantic relationships, and friendships to verify that the away teams satisfy the following rules.

- If two people are best friends and they both participate in exploring the planets, they must be in the same team. Note that this rule does *not* apply to the situation when a person is sent to explore a planet while her/his best friend remains on the ship, which is perfectly acceptable.
- A person and her/his former spouse must not be on the same planet; however, this rule does *not* apply if a person and her/his former spouse are now best friends.
- A person and her/his former romantic partner must not be in the same team; however, this rule does *not* apply in the following two cases:
  - She/he and her/his former romantic partner are now best friends.
  - She/he is now married and her/his former romantic partner is also married.
- A group leader cannot have her/his spouse or romantic partner in her/his group.
- A planetary coordinator cannot have her/his spouse or romantic partner on the same planet.

Since manual verification of a team roster against these rules is tedious and error-prone, Troy wants to automate it, and your task is to implement a program for this task. You may assume that the number of crew members is at most 1000 and that their relationships satisfy the following constraints.

- The friendships, marriages, and romantic relationships are symmetric; for instance, if Deanne is Will's romantic partner, then Will is Deanne's partner.
- A married person has one spouse and no other romantic partners.
- An unmarried person has at most one romantic partner; some people may have no romantic partners.
- If two people are now married, they cannot be "formerly married" to each other at the same time. If two people were married once, then divorced, and then have married each other again, they are *not* considered formerly married to each other.

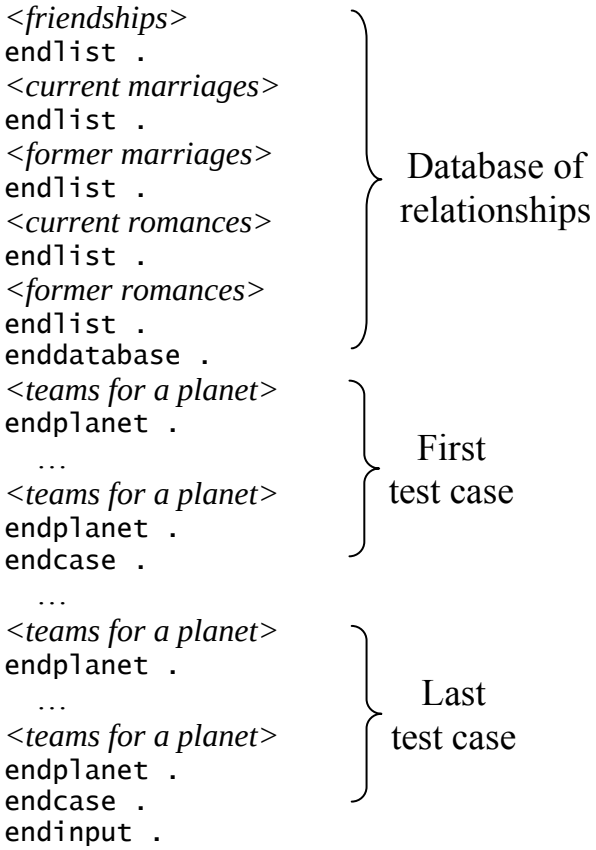
- A person's romantic partner cannot be her/his "former romantic partner" at the same time. If two people were romantic partners in the past, then broke up, and then have gotten together again, they are *not* considered former romantic partners.
- A person has at most one best friend; some people may have no friends.
- A person never marries or dates her/his best friend; however, a person may be the best friend of her/his *former* spouse or romantic partner.

Note that you *cannot* assume that the two people in a marriage or a relationship always have opposite genders. You cannot even assume that every crew member has a gender, since many of them come from alien species.

Input

The input includes a database of social relationships followed by multiple test cases, which correspond to different away-team compositions. The input structure is more complex than in other problems, and its description consists of three sections. The first section is an explanation of the overall high-level structure of the input, the second is the syntax of a relationship database, and the third is a specification of test cases.

High-level structure: The first part of the input is a relationship database, which comprises five lists: friendships, current marriages, former marriages, current romances, and former romances. The rest of the input is a set of test cases, where each case specifies away teams and their distribution among planets; the number of cases is at most 20. The overall input structure is as follows:



Relationships: The database of relationships consists of five lists (see above), where each list has the following format:

<*name*> <*name*>
<*name*> <*name*>
…
<*name*> <*name*>

That is, a relationship list is a set of specific relationships, each on a separate line. The description of a relationship consists of two crew-member names, separated by a single space. We represent names by lower-case alphabetical strings, with length between 1 and 20 characters, and we assume that each crew member has a unique name.

The number of relationships in a list is between 1 and 100000, and all listed relationships are distinct. Note that, while a list cannot include the same pair of names twice, different relationship lists may include the same pair. For instance, if Mary and John were romantic parnters, then got married, then divorced, and later have become best friends, then the pair "`mary john`" (or "`john mary`") is in three lists: friendships, former marriages, and former romances.

Test cases: A test case includes away-team rosters for multiple planets (see above); the number of planets is between 1 and 100. For each planet, the roster of teams sent to that planet has the following format:

<*coordinator-name*> <*name*> ... <*name*>
<*leader-name*> <*name*> ... <*name*>
…
<*leader-name*> <*name*> ... <*name*>

That is, the roster for a planet is a set of teams, one team per line; the number of teams sent to a planet is between 1 and 10. A team description comprises crew-member names, separated by single spaces; the number of people in a team is between 2 and 10. The team member listed first is the team leader; the leader of the first team is the planetary coordinator. Note that a person may be on only one team, which means that each name occurs at most once within a test case.

Output

For each test case, the output is a single line. If the away teams satisfy all Deanna's rules, the output is "`approved`"; else, it is "`need changes`".

Sample input

```
jeanluc will
geordi miles
tpol trip
mary john
endlist .
keiko miles
beverly jeanluc
endlist .
mary john
endlist .
deanna will
endlist .
deanna worf
jenna data
lisa trip
tpol trip
mary john
endlist .
enddatabase .
jeanluc will deanna
geordi miles keiko
endplanet .
tpol trip beverly
worf data lisa
mary john
endplanet .
endcase .
jeanluc geordi beverly
miles will keiko
endplanet .
endcase .
data jenna
endplanet .
jeanluc deanna worf
endplanet .
endcase .
endinput .
```

Sample output

```
approved
need changes
need changes
```

Problem I: Worf

Lieutenant Commander Worf (Klingon) was born in the Klingon Empire, which is a nation of warriors, with culture focused on the military honor and glory. Although Worf grew up in a human family and became a Federation citizen, he has accepted many values of his original homeland, including Klingon pastime activities.

In particular, he likes to play *catch-a-targ*; it is an ancient Klingon game involving two warriors and one *targ*, which is a Klingon pet animal somewhat similar to a boar (see the picture). The two warriors play as a team, trying to catch the targ, while the frightened animal does its best to avoid the capture as long as possible.

Unfortunately, the warriors sometime fail to capture the targ and have to admit their defeat, which is a great dishonor. The thoughts of this possibility cause significant stress and make the game much less fun. To avoid this problem, Worf has suggested using a computer to predict the game outcome. If a computer simulation shows that the warriors will capture the targ, they can enjoy the game without fearing for their honor. If not, they need to play it on a different terrain that would simplify their task.

After thinking about it, Worf has decided to represent the terrain by an unweighted undirected graph, which may or may not be connected, where the warriors and the targ occupy specific nodes and move from node to node along edges (see the pictures on the next page). The targ makes the first move; it may stay in its current node or run to one of the adjacent nodes. Then, the warriors make their moves, which are similar to the targ's move; that is, each warrior may remain in place or run to an adjacent node. Then, the targ makes its next move, the warriors make their moves, and so on. Note that the warriors may sometime share a node and sometime occupy different nodes. If one of them reaches the targ's node, he or she captures the animal, thus winning the game. If the warriors cannot catch the targ in an hour, they admit their defeat. The game duration in seconds equals the number of the targ's moves, which means that the warriors have to capture it within 3600 moves.

Your task is to implement a program that predicts the game outcome, assuming that the targ and the warriors use the optimal strategy.

Input

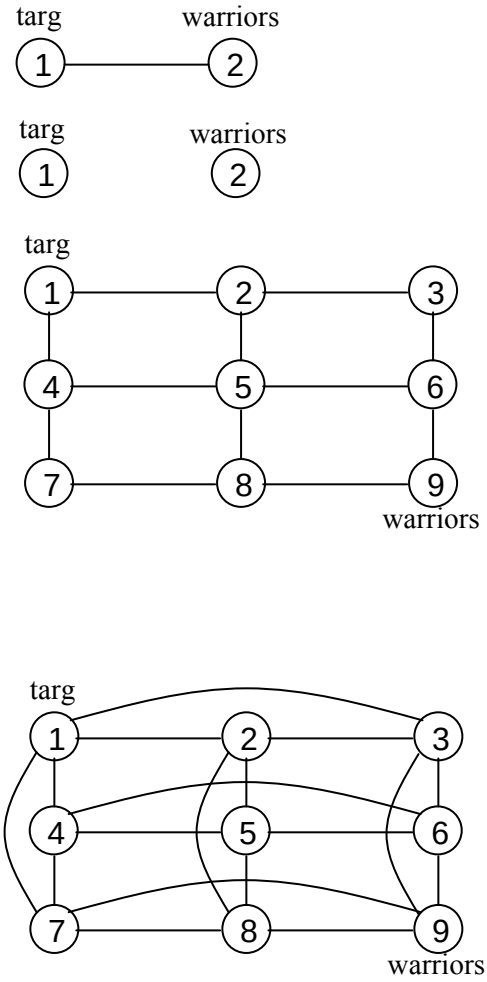
The input includes multiple test cases, which correspond to different terrains for playing the game; the number of cases is at most 20. The first line of a test case is an integer n between 2 and 50, which is the number of nodes in the graph. We assume that the nodes are numbered from 1 to n, the targ is initially in node 1, and both warriors are initially in node n. The other lines are distinct pairs of integers between 1 and n, which represent the edges. The integers in a pair are separated by a single space, and the first is strictly smaller than the second. Since all pairs are distinct, their number is at most $n \cdot (n - 1) / 2$. The last line of a test case is "`0 0`" (two zeros, separated by a single space), which does not represent an edge. The last line of the input, after the last test case, is "`–1 –1`".

Output

For each test case, the output is an integer on a separate line, which is the duration of an optimally played game. If the warriors capture the targ, the output is the number of the targ's moves before the capture, which is between 1 and 3600; else, the output is 3601.

Sample input

```
2
1 2
0 0
2
0 0
9
1 2
1 4
2 3
2 5
3 6
4 5
4 7
5 6
5 8
6 9
7 8
8 9
0 0
9
1 2
1 3
1 4
1 7
2 3
2 5
2 8
3 6
3 9
4 5
4 6
4 7
5 6
5 8
6 9
7 8
7 9
8 9
0 0
−1 −1
```



Sample output

```
1
3601
4
3601
```

Problem J: Q

Q is *not* a member of the *Enterprise* crew. He is a mysterious near-omnipotent being, capable of affecting the universe on a cosmic scale, who occasionally entertains himself by toying with the *Enterprise* crew. He would show up unexpectedly on the *Enterprise* and give Captain Picard a tough test, sometimes threatening to destroy the humankind if the captain fails. Since Picard has passed all tests so far, it is unclear whether Q would actually carry out his threat in case of a failure.

Recently, Q has learned about NP-completeness and has understood that the Subset Sum Problem is NP-hard, which seems to imply that any program for solving it would be impractically slow. To put his new knowledge to use, he has tasked Picard and his crew with writing an efficient Subset Sum implementation and, as usual, has threatened to wipe out the humankind if they do not produce it by the end of the day. Specifically, Picard has to deliver a program that inputs a set of integers, which may include up to 40 elements, and determines whether it has a subset that sums to zero.

Luckily for the humans, Q is not strong in algorithm theory and has not realized that his bound on the input size allows an efficient solution. Less luckily, the *Enterprise* crew does not include any algorithms experts who can come up with an efficient implementation. Your task is to help Picard by implementing a solution to Q's problem.

Input

The input includes multiple test cases, which represent different instances of the Subset Sum Problem; the number of cases is at most 20. A test case is a set of distinct nonzero integers between −50000000 and 50000000, each on a separate line; the number of integers in a test case is between 2 and 40. The last line of a test case is "0" (zero), which is not an element of the given set. The last line of the input, after the last test case, is `239239239`.

Output

For each test case, the output is a single line. If a given set of integers has a nonempty subset that sums to zero, the output is "`yes`"; else, it is "`no`".

Sample input

```
1000000
–1000000
0
2
4
10
–5
–9
0
1
2
3
4
–11
0
1
2
4
8
16
32
64
–128
–256
–512
–1024
0
239239239
```

Sample output

```
yes
yes
no
no
```