2012 Mid-Atlantic Regional Programming Contest

Welcome to the 2012 ICPC Mid-Atlantic Regional. Before you start the contest, please take the time to review the following:

The Contest

1. There are eight (8) problems in the packet, using letters A–H. These problems are NOT sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Fifty Coats of Gray	Yellow
B	Component Testing	Green
C	Collision Detection	Silver
D	The Dueling Philosophers Problem	Black
E	Party Games	Pink
F	Funhouse	Orange
G	A Terribly Grimm Problem	Purple
H	Tsunami	Red

2. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Yes	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Incomplete Output	Your submission did not produce all of the required output.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.
Other-Contact Staff	Contact your local site judge for clarification.

3. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is

solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

4. This problem set contains sample input and output for each problem. However, the judges will test your submission against longer and more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive a judgment stating that your submission was incorrect, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification.

   If a clarification is issued during the contest, it will be broadcast to *all* teams.

   If the judges believe that the problem statement is sufficiently clear, you will receive the response, "No response, read problem statement." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found.

   You may *not* submit clarification requests asking for the correct output for inputs that you provide, e.g., "What would the correct output be for the input ...?" Determining that is your job unless the problem description is truly ambiguous. Sample inputs *may* be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: ..., would both this: ... and this: ... be valid outputs?".

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for problem B followed by a run for problem C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the local site judge to determine the cause of the delay.** Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

8. The submission of code deliberately designed to delay, crash, or otherwise negatively affect the contest itself will be considered grounds for immediate disqualification.

Your Programs

9. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error (cerr in C++ or System.err in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

   ```
   program < file.in
   ```

10. Unless otherwise specified, all lines of program output

    - must be left justified, with no leading blank spaces prior to the first non-blank character on that line,
    - must end with the appropriate line terminator (`\n`, `endl`, or `println()`), and
    - must not contain any blank characters at the end of the line, between the final specified output and the line terminator.

    You must not print extra lines of output, even if empty, that are not specifically required by the problem statement.

11. Unless otherwise specified, all numbers in your output should begin with the minus sign (-) if negative, followed immediately by 1 or more decimal digits. If the number being printed is a floating point number, then the decimal point should appear, followed by the appropriate number of decimal digits. For output of real numbers, the number of digits after the decimal point will be specified in the problem description (as the "precision").

    All floating point numbers printed to a given precision should be rounded to the nearest value. For example, if 2 decimal digits of precision is requested, then 0.0152 would be printed as "0.02" but 0.0149 would be printed as "0.01".

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure that you use a printing technique that rounds to the appropriate precision.

12. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

13. All lines of program input will end with the appropriate line terminator (e.g., a linefeed on Unix/Linux systems, a carriage return-linefeed pair on Windows systems).

14. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

15. Every effort has been made to ensure that the compilers and run-time environments used by the judges are as similar as possible to those that you will use in developing your code. With that said, some differences may exist. It is, in general, your responsibility to write your code in a portable manner compliant with the rules and standards of the programming language. You should not rely upon undocumented and non-standard behaviors.

    (a) One place where differences are likely to arise is in the size of the various numeric types. Many problems will specify minimum and maximum values for numeric inputs and outputs. You should write your code with the understanding that, on the *judges'* machines:

    - A C++ `int`, a C++ `long`, and a Java `int` are all 32-bits wide.
    - A C++ `long long` and a Java `long` are 64-bits wide.
    - A `float` in both languages is a 32-bit value capable of holding 6-7 decimal digits, though many library functions will be less accurate.
    - A `double` in both languages is a 64-bit value capable of holding 15-16 decimal digits, though many library functions will be less accurate.

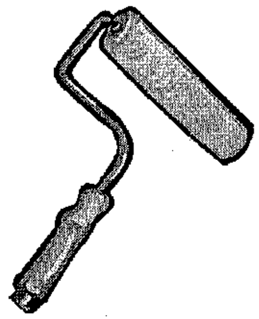
    The data types on your own machines may differ in size from these, but if you follow the guidelines above in choosing the types to hold your numbers, you can be assured that they will suffice to hold those values on the judges' machines.

    (b) Another common source of non-portability is in C/C++ library structures. Although the C & C++ standards are very explicit about which header file must declare certain std symbols, the standards do not prohibit other headers from duplicating or loading extra symbols.

    For example, if your program uses both `cout` and `ifstream`, you might find that your code compiles if you only `#include <fstream>`, because, as it happens, on *your* machine the `fstream` header `#includes` the `iostream` header where `cout` is properly declared. However, you cannot rely upon the judges' machines having libraries with the same structure. So it is *your* responsibility to `#include` the appropriate headers for whatever std library features you use.

Good luck, and HAVE FUN!!!

Problem A: Fifty Coats of Gray



A contractor is planning to bid on interior painting for an apartment. These apartments are for student housing, so they are to be single-room efficiencies and have basic drywall walls and ceilings, with no particular architectural features like crown molding. He would like to find a quicker way to estimate how much paint it will take to paint the walls and ceilings for each job. The plan for these buildings is to paint the four walls and the ceiling. Of course, no paint is needed for window and door openings. All rooms, windows and doors are rectangular. All rooms will be painted the same color.

The contractor will provide you with information about the dimensions of the rooms, the windows and doors for each floor plan, and the number of apartments. Your team is to write a program that will tell him how many cans of paint he should include in his bid.

Input

There will be several test cases in the input. Each test case begins with a line with 6 integers:

n width length height area m

where n ($1 \leq n \leq 100$) is the number of apartments, width ($8 \leq$ width ≤ 100) is the width of each room, length ($8 \leq$ length ≤ 100) is the length of each room, height ($8 \leq$ height ≤ 30) is the height of each room, area ($100 \leq$ area $\leq 1,000$) is the area in square feet that can be covered by each can of paint, and m ($0 \leq m \leq 10$) is the number of windows and doors.

On each of the next m lines will be two positive integers, width and height, describing a door or window. No window or door will be larger than the largest wall. All linear measures will be expressed in feet. The input will end with a line with six 0s.

Output

For each test case, output a single integer on its own line, indicating the number of cans of paint needed to paint all of the walls and ceilings of all of the apartments.

Example

Given the input

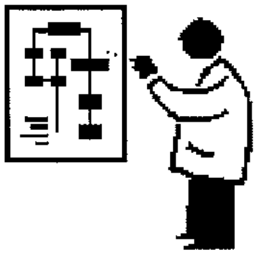
```
50 8 20 8 350 2
6 3
3 3
50 8 20 8 300 3
6 3
5 3
3 3
```

```
0 0 0 0 0 0
```

the output would be

```
83
95
```

Problem B: Component Testing



The engineers at ACM Corp. have just developed some new components. They plan to spend the next two months thoroughly reviewing and testing these new components. The components are categorized into several different classes, depending on their complexity and importance. Components in different classes may require different number of reviewers, whereas components in the same class always require the same number of reviewers.

There are also several different job titles at ACM Corp. Each engineer holds a single job title. All engineers holding a given job title have the same limit on the number of components that they can review. Note that an engineer can be assigned to review any collection of components and will be able to complete the task, regardless of which classes the components belong to. An engineer may review some components of the same class, and others from different classes, but an engineer cannot review the same component more than once.

Can the engineers complete their goal and finish testing all components in two months?

Input

There will be multiple test cases in the input.

The first line of each test case contains two integers n ($1 \leq n \leq 10,000$) and m ($1 \leq m \leq 10,000$), where n is the number of component classes and m is the number of engineer job titles.

Each of the next n lines contains two integers j ($1 \leq j \leq 100,000$) and c ($0 \leq c \leq 100,000$), indicating that there are j components in this class and that each component requires at least c different reviewers.

Then each of the next m lines each contains two integers k ($1 \leq k \leq 100,000$) and d ($0 \leq d \leq 100,000$), indicating that there are k engineers with this job title and that each engineer may be assigned to review at most d components.

The input will end with a line with two 0s.

Output

For each test case, print a single line containing 1 if it is possible for the engineers to finish testing all of the components and 0 otherwise.

Example

Given the input

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

```
5 2
1 1
1 3
1 1
1 3
1 1
1 20
1 4
0 0
```

the output would be

```
1
0
```