How to Avoid Disqualification in 75 Easy Steps (avoid)

You are standing in front of the open safe, a medal in your hand. But your triumph turns to despair as you look around: a message on a tie you found in the room reveals that your assistant exposed your plan to the Scientific Committee! Now the two committee chairmen are hiding in the building to stop you from escaping...

Luckily, you have R vacuum cleaner robots left over from your trade with your fellow contestants. You want to use these robots to locate the two chairmen so that you can avoid them during your escape. You can instruct each robot to scout several of 1 000 positions where the chairmen could be located. Unfortunately, the software of these robots is quite simple.* Each robot can only detect whether or not *there is at least one chairman at its scouted positions*.

To make things worse, every robot needs a full hour to scout its positions before returning with its result to you. Because this will drain the robot's battery, *you can send out each robot only once*.

As you don't want to be late for the evening activities, you want to know the positions of the chairmen *after at most H hours*. In particular, you might be forced to send out several robots at once without waiting for the previous robots to return. You can assume that the two chairmen stay at the same positions all the time.†

Write a program which plans this scouting mission and determines where the two chairmen are located.

Communication

This is a communication task. You must implement the function **pair**⟨**int**, **int**⟩ *scout*(**int** R, **int** H) where R and H are as described above. For each testcase, this function is called exactly once and should return a pair of two integers $1 \le a, b \le 1\,000$ ($a = b$ is allowed), the positions of the two chairmen. Inside *scout*, you can use the following other functions provided by the grader:

- **void** *send*(**vector**⟨**int**⟩ P) sends a robot to scout the positions $P[0], \ldots, P[k-1]$ (where k is the length of the array P). The positions $P[i]$ must be pairwise distinct integers between 1 and 1 000. You can call this function at most R times per testcase.
- **vector**⟨**int**⟩ *wait*() waits an hour. This function returns an array with exactly one entry for each robot sent out one hour ago (by a call to *send* after the previous call to *wait* or after the beginning of the program). The entry at index i is `1` if the $(i+1)$-th of these robots has detected at least one chairman at its scouted positions, and `0` otherwise. You can call this function at most H times per testcase.

If any of your function calls does not satisfy the above constraints, your program will be immediately terminated and judged as **Not correct** for the respective testcase. You must not write to standard output or read from standard input; otherwise, you may receive the verdict **Security violation!**. You are however free to write to the standard error stream (*stderr*).

You must include the file `avoid.h` in your source code. To test your program locally, you can link it with `sample_grader.cpp`, which can be found in the attachment for this task in CMS. See below for a description of the sample grader, and see `sample_grader.cpp` for instructions on how to run it with your program. The attachment also contains a sample implementation as `avoid_sample.cpp`.

* After all, you sold all the fancy ones to the other contestants!
† Due to the late-night task preparations they are simply too exhausted to move.

Constraints and grading

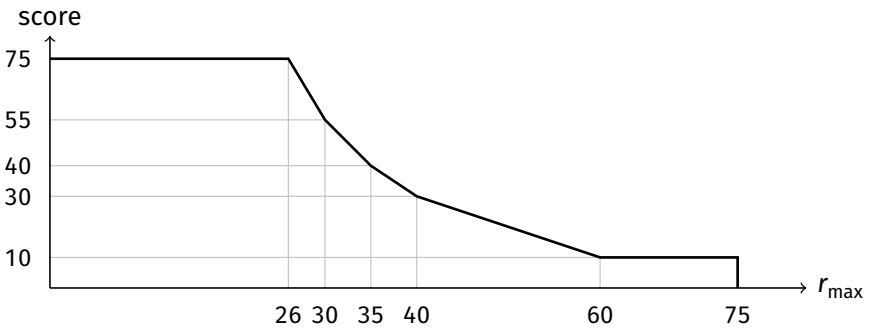
Subtask 1 (10 points). $R = 10$, $H = 1$ and both chairmen are located at the same position.

Subtask 2 (5 points). $R = H = 20$

Subtask 3 (10 points). $R = 30$, $H = 2$

Subtask 4 (75 points). $R = 75$, $H = 1$

Partial scoring. In Subtask 4, your actual score depends on the maximum number $r_{\max}$ of robots sent out over all testcases in this subtask according to the following piecewise linear function:



In particular, to get full score you must not make more than 26 calls to *send* per testcase in the last subtask. You can also find a table listing all the individual scores in the attachments for this task as `score_table.txt`.

Example interaction

Consider a testcase with $R = 75$ and $H = 20$ where the chairmen are located at positions 13 and 37. First, the grader calls your function *scout* as *scout*(75, 20). Then, an interaction between your program and the grader could look as follows:

Your program	Return value	Explanation
send({42, 13, 37})	—	send a robot to positions 13, 37 and 42
send({47, 11})	—	send a robot to positions 11 and 47
wait()	{1, 0}	wait an hour for the return of the robots; only the first robot has detected a chairman
send({42})	—	send a robot to position 42
wait()	{0}	wait an hour; there is no chairman at position 42
return {13, 37}	—	you are convinced that the chairmen are located at positions 13 and 37 the solution is correct and is accepted

Returning the pair {37, 13} would be accepted as well. Note that the above queries are of course not sufficient to determine the positions of the chairmen with certainty: For example, both being at position 37 or one chairman being at position 13 while the other is at position 100 would also be consistent with all answers to *wait*, so the grader could also have rejected this solution.

The above interaction is reproduced by `avoid_sample.cpp` on the public testcase.

Grader

The sample grader first expects on standard input the integers R and H and the positions a and b of the chairmen ($1 \le a, b \le 1\,000$). Then, the grader calls *scout*(R, H) and writes to standard output a protocol of all grader functions called by your program. Upon termination, it writes one of the following messages to standard output:

Invalid input. The input to the grader via standard input was not of the above format.

Invalid send. You called *send* with invalid parameters.

Out of robots. You called *send* more than R times.

Out of time. You called *wait* more than H times.

Wrong answer. The positions returned by *scout* are not the positions of the chairmen.

Correct: *r* robot(s) used, *h* hour(s) passed. The positions returned by *scout* are the positions of the chairmen, there were r calls to *send*, and there were h calls to *wait*.

In contrast, the grader actually used to judge your program will only output **Not correct** (for any of the above errors), **Security violation!**, or **Correct: *r* robot(s) used, *h* hour(s) passed**. Moreover, the grader is *adaptive*, i.e. the positions of the chairmen may depend on the behavior of your program in the current as well as in earlier runs. Both the sample grader and the grader used to judge your program will terminate your program automatically whenever one of the above errors occurs.

Limits

Time: 0.25 s
Memory: 512 MiB