Task 3. Stone

⌛0.1 sec. 💾 256 MB

You are Bobi and you love shiny things. After you found the minimum spanning tree in your graph, under the tree you found a shiny stone which you decided to rub, because you are Bobi and you love shiny things. The stone, however, was shiny for a reason and managed to fling you into an N-dimensional universe. You are starting to get hungry, therefore you want to return to your universe of origin as soon as possible, and luckily for you, at one of the points in this N-dimensional world there is a portal which will bring you back. Unluckily for you, you do not know where it is. You have at your disposal a drone which can move in space. It uses one percent of its battery for every move, but, since it is old, it charges only up to 12 percent. You receive it fully charged according to its capabilities. You can charge it at any time, but for that it has to be at the point at which you are as well. You and the drone start the journey at the same point. You are getting hungrier and hungrier, therefore you have to think quickly about how to get out of the N-dimensional universe.

Implementation details

This task is interactive. The function `void solve(long long n)`, which you have to write, will be called only once by the program of the jury and will receive as an argument the integer N.

`void solve(long long N, long long M);` – the function which you have to implement. The program of the jury calls it with a parameter N – the number of dimensions and M – the maximum position. By the end of the function you must have called the function `portal()`.

`std::vector <long long> query();` – You can call this function. It returns a vector of N elements, each describing the position of the portal relative to the position of the drone. If element i of the vector is equal to 1, the coordinate in the i-th dimension of the portal is greater than the coordinate in the i-th dimension of the drone. Respectively, if it is equal to 0, it is smaller, and if it is 2, it is equal.

`void moveself(std::vector <long long> pos);` – You can call this function. It moves you to the position described in the vector *pos*.

`void movedrone(std::vector <long long> pos);` – You can call this function. It moves the drone to the position described in the vector pos. Do not forget that one call of this function costs you one percent of the drone's battery.

`void chargedrone();` – You can call this function. It charges the drone fully, i.e. up to 12 percent. If you and the drone are not at the same point, the program terminates and you will get Wrong Answer.

`void portal();` – You can call this function. It terminates your program. It must be called at the end of the function `solve();`.

`std::vector <long long> getposself();` – You can call this function. It returns your current position.

`std::vector <long long> getposdrone();` – You can call this function. It returns the current position of the drone.

`void changename(string name);` – You can call this function. It changes your name to name.

`void hugdrone();` – You can call this function. Through it you hug the drone, because you are hungry and you are sad when you are hungry, so you want to hug someone (or something). If you and the drone are not at the same point, the program terminates and you get Wrong Answer.

Your program `stone.cpp` must implement the function `solve`. It may also contain other code and functions necessary for your work, but it must not contain the main function `main`. Also, you must

not read from the standard input or print to the standard output. Your program must include the header file `stone.h` through a directive to the preprocessor:

```
#include "stone.h"
```

Constraints

- $1 \leq N \leq 10^5$

Subtasks

Subtask	Points	Constraints
1	25	$1 \leq N \leq 1000$
2	75	$1 \leq N \leq 10^5$

The points for a given subtask are awarded only if all the tests provided for it are passed successfully.

Scoring

If you have used more than the optimal number of queries, you get 50% of the points for the respective test.

Local testing

You are provided with the file `Lgrader.cpp`, which you can compile together with your program in order to test it. On startup the program will read from the standard input the values of N and M. After that the communication which takes place will be printed. You may modify the provided file as you wish.