Chika Wants to Cheat

Problem Name	Cheat
Input File	Interactive task
Output File	Interactive task
Time limit	2 seconds
Memory limit	512 megabytes

Chika has a deck of q playing cards numbered with various positive integers. She wants to play some games with her friends from the Shuchi'in Academy student council using these cards, but she also wants to win, so she decides to secretly mark the back of the cards in her deck.

The cards are all square-shaped and of size 2×2, where the bottom-left corner has coordinates $(0, 0)$ and the top-right corner has coordinates $(2, 2)$. Chika draws a certain pattern on the back of each card, so that she will later know, by looking at the pattern, which number is written on the front of the card. She draws such a pattern using the following procedure: As many times as she wants (possibly 0 times), she picks two distinct points A and B that have integer coordinates relative to the bottom-left corner of the card and draws a **straight line segment** between them.

Chika will only draw **valid** segments, that is, segments between two points A and B for which there is no another point C (distinct from A and B) with integer coordinates that also lies on the segment. For example, the segment between $(0, 0)$ and $(2, 2)$ is **not valid** as it contains point $(1, 1)$, but segments between $(0, 0)$ and $(1, 1)$ and between $(1, 1)$ and $(2, 2)$ are both **valid**, and Chika can even draw both of them on the same pattern. Also, note that the segments have no direction: A segment drawn from A to B is **identical** to both itself and the segment drawn in the reverse direction, from B to A.

Importantly, Chika wants to make sure that she will recognize her cards regardless of how they are rotated. A card can be rotated 0, 90, 180 or 270 degrees counter-clockwise with respect to the original orientation.

Your task is to help Chika design the patterns for the q cards in her deck and then recognize those cards later on.

Implementation

This is an interactive task with two stages, ***<u>each stage involving a separate run of your program</u>***. You need to implement two functions:

- A `BuildPattern` function that returns the pattern to be drawn on the back of a given card. This function will be called q times in the first stage.
- A `GetCardNumber` function that returns the number of a (possibly rotated) card that carries a given pattern drawn in the first stage. This function will be called q times in the second stage.

The first function

```
std::vector<std::pair<std::pair<int, int>, std::pair<int, int>>> BuildPattern(int n);
```

takes a single parameter n, the number that is written on the front of the card. You need to return a `std::vector` containing the segments that Chika draws as a pattern on the back of the card to recognize it later. A segment is represented as a `std::pair` of points, and a point is represented as a `std::pair` (x, y) of integer coordinates relative to the bottom-left corner of the card, where $0 \leq x, y \leq 2$. All segments that Chika draws need to be valid and pairwise non-identical. It is guaranteed that all q calls to BuildPattern receive different values for the parameter n.

After receiving all the patterns for the q cards, the grader can do any of the following operations, any number of times, on each of the patterns:

- Rotate the entire pattern by 0, 90, 180 or 270 degrees counter-clockwise.
- Modify the order of segments in the `std::vector` representation of the pattern.
- Change the order of the endpoints of a segment in the pattern. (A segment drawn from A to B may become the identical segment from B to A.)

The second function,

```
int GetCardNumber(std::vector<std::pair<std::pair<int, int>, std::pair<int, int>>> p);
```

takes a single parameter p, a `std::vector` of segments describing the pattern that is drawn by Chika on the back of the card, based on the return value of an earlier call to your `BuildPattern` function. The function must return the number n written on the front of the card. Recall that the pattern p is not necessarily in the original form that is returned by `BuildPattern`; it may have been subject to the three operations mentioned above. It is also possible that the order of the cards is different from the order they were given in the first stage, but it is guaranteed that each card will be used exactly once.

Constraints

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ for all calls to function `BuildPattern`.
- Note that there exist algorithms to construct patterns such that one can recognize $67\,000\,000$ different cards.

Scoring

- Subtask 1 (2 points): $n \leq 2$.
- Subtask 2 (9 points): $n \leq 25$.
- Subtask 3 (15 points): $n \leq 1\,000$ and the grader will **not rotate** the patterns between stages 1 and 2. (The grader **may** perform the other two operations.)
- Subtask 4 (3 points): $n \leq 16\,000\,000$ and the grader will **not rotate** the patterns between stages 1 and 2. (The grader **may** perform the other two operations.)
- Subtask 5 (24 points): $n \leq 16\,000\,000$.
- Subtask 6 (18 points): $n \leq 40\,000\,000$.
- Subtask 7 (29 points): No further constraints.

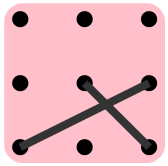
Sample Interaction

Function call	Return value	Explanation
First stage begins.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	We have to create a pattern for number 3 on the 2×2 sized card. We decide to draw 2 segments: - between $(0, 0)$ and $(2, 1)$, - between $(1, 1)$ and $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	We have to create a pattern for number 1 on the 2×2 sized card. We decide to draw 1 segment: - between $(0, 1)$ and $(0, 0)$.
First stage ends.	-	-
Second stage begins.	-	-
`GetCardNumber( {{{0, 0}, {0, 1}}})`	`1`	We get a pattern made up of 1 segment: - between $(0, 0)$ and $(0, 1)$. This is the same pattern as we would get from drawing the segment: - between $(0, 1)$ and $(0, 0)$. which is exactly the same pattern with the same orientation (rotated by 0 degrees) we returned on the second

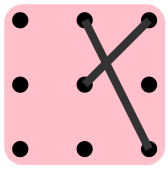
		call to function `BuildPattern`. Therefore, we return 1.
`GetCardNumber({{{1, 1}, {2, 2}}, {{1, 2}, {2, 0}}})`	`3`	We get a pattern made up of 2 segments: - between $(1, 1)$ and $(2, 2)$, - between $(1, 2)$ and $(2, 0)$. This is the pattern we returned on the first call to the `BuildPattern` function, rotated by 90 degrees counter-clockwise. Therefore, we return 3.
Second stage ends.	-	-

The next three pictures represent, in order:

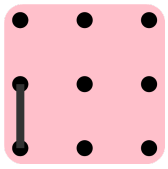
- The pattern returned as output by the first call to `BuildPattern`:



- The pattern received as parameter by the second call to `GetCardNumber`, which is the first pattern after it is rotated by 90 degrees counter-clockwise.



- The pattern returned as output by the second call to `BuildPattern`, which is also the same pattern received as argument by the first call to `GetCardNumber`.



Sample Grader

The provided sample grader, `grader.cpp`, in the task attachment `Cheat.zip`, reads an integer q from standard input and then will do the following steps q times:

- Read an integer n from standard input.
- Call `BuildPattern(n)` and store the return value in a variable p.
- Call `GetCardNumber(p)` and print the return value to standard output.

You can modify your grader locally if you wish to do so.

To compile the sample grader with your solution, you may use the following command at the terminal prompt:

```
g++ -std=gnu++11 -O2 -o solution grader.cpp solution.cpp
```

where `solution.cpp` is your solution file to be submitted to CMS. To run the program with the sample input provided in the attachment, type the following command in the terminal prompt:

```
./solution < input.txt
```

Please note that, unlike the sample grader, the real grader on CMS will perform the first stage and second stage in separate executions of your program.