Chika Wants to Cheat

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

У Чики есть колода из q карт, на лицевой стороне каждой карты нанесено целое положительное число. Она хочет играть этой колодой в разные игры со своими друзьями из академии Шучи'ин, но при этом она хочет победить, поэтому она решила нанести секретные пометки на обратные стороны карт своей колоды.

Каждая карта представляет собой квадрат размером 2×2, где левый нижний угол имеет координаты $(0, 0)$, а правый верхний имеет координаты $(2, 2)$. Чика планирует нарисовать на обратной стороне каждой карты определенный узор, таким образом позже она сможет понять, посмотрев на этот узор, какое число написано на лицевой стороне карты. Она рисует узор, используя следующую процедуру: произвольное количество раз (возможно 0), она выбирает две различные точки A и B с целыми координатами (если отсчитывать от левого нижнего угла карты) и проводит **отрезок** между ними.

При этом Чика проводит только **корректные** отрезки, а именно, такие отрезки, для которых между точками A и B не существует точки C (отличной от A и B) с целыми координатами, которая также лежит на этом отрезке. Например, отрезок между точками $(0, 0)$ и $(2, 2)$ является **некорректным**, поскольку на нем лежит точка $(1, 1)$, но отрезки между точками $(0, 0)$ и $(1, 1)$, а также между точками $(1, 1)$ и $(2, 2)$ оба **корректные**, в частности Чика может провести оба этих отрезка. Отрезки не направленные, отрезок из точки A в точку B **совпадает** с отрезком, проведенным в противоположном направлении, из B в A.

Важно, что Чика хочет нанести такие узоры, чтобы она могла узнать карты даже если они будут повернуты. Карта может быть повернута на 0, 90, 180 и 270 градусов против часовой стрелки относительно своей исходной ориентации.

Ваша задача помочь Чике придумать узоры для всех q карт, чтобы позже она могла узнать каждую из них.

Implementation

Это интерактивная задача с двойным запуском, ***<u>ваша программа при проверке будет запущена дважды</u>***. Вам необходимо реализовать две функции:

- Функция `BuildPattern` должна возвращать узор, который будет изображен на обратной стороне карты. Эта функция будет вызвана q раз при первом запуске вашей программы.
- Функция `GetCardNumber` должна опознавать карту (возможно, повернутую) с узором, который был изображен на ней во время первого запуска программы. Эта функция будет вызвана q раз при втором запуске программы.

Первая функция

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

принимает одно число n — число, написанное на лицевой стороне карты. Функция должна вернуть `std::vector`, содержащий отрезки, которые Чика проведет на обратной стороне карты, чтобы потом опознать ее. Отрезок задается как `std::pair` точек, а точка задается как `std::pair` (x, y) целочисленных координат относительно левого нижнего угла карты, $0 \le x, y \le 2$. Все отрезки, которые проводит Чика, должны быть корректными и попарно различными. Гарантируется, что все q вызовов `BuildPattern` получат различные значения параметра n.

После того, как узоры для всех q карт будут сформированы, грейдер проделает следующие операции произвольное число раз с заданными узорами:

- Повернуть весь узор на 0, 90, 180 или 270 градусов против часовой стрелки.
- Изменить порядок, в котором отрезки следуют в `std::vector`, задающем узор.
- Изменить порядок в котором задаются концы отрезка. (Заменить отрезок из A в B на отрезок из B в A.)

Вторая функция,

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

принимает один аргумент p, `std::vector` отрезков, которые задают узор, изображенный на обратной стороне карты, а именно преобразованный описанным выше способом результат выполнения функции `BuildPattern`. Функция должна вернуть число n, написанное на лицевой стороне карты. Обратите внимание, что p не обязательно содержит результат выполнения `BuildPattern`; к нему могли быть применены три описанные выше операции. Кроме того, порядок, в котором карты будут переданы в функцию при втором запуске может

отличаться от порядка, в котором они были переданы при первом запуске, но гарантируется, что каждая карта будет использована в точности один раз.

Constraints

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ во всех вызовах `BuildPattern`.
- Существует алгоритм, который формирует узоры, которые позволяют различить $67\,000\,000$ карт.

Scoring

- Subtask 1 (2 points): $n \leq 2$.
- Subtask 2 (9 points): $n \leq 25$.
- Subtask 3 (15 points): $n \leq 1\,000$ грейдер **не поворачивает** узоры между запусками 1 и 2. (Грейдер при этом **может** выполнять две оставшиеся операции.)
- Subtask 4 (3 points): $n \leq 16\,000\,000$, грейдер **не поворачивает** узоры между запусками 1 и 2. (Грейдер при этом **может** выполнять две оставшиеся операции.)
- Subtask 5 (24 points): $n \leq 16\,000\,000$.
- Subtask 6 (18 points): $n \leq 40\,000\,000$.
- Subtask 7 (29 points): Дополнительных ограничений нет.

Sample Interaction

Вызов функции	Возвращаемое значение	Пояснение
Первый запуск.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Необходимо создать узор для числа 3 на карте 2×2. Чика решила провести 2 отрезка: - между $(0, 0)$ и $(2, 1)$, - между $(1, 1)$ и $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Необходимо создать узор для числа 1 на карте 2×2. Чика решила провести 1 отрезок: - между $(0, 1)$ и $(0, 0)$.
Первый запуск завершается.	-	-
Начинается второй запуск.	-	-
`GetCardNumber(`	`1`	Передан узор из 1 отрезка:

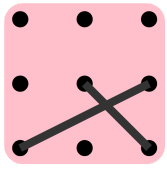
`{{{0, 0}, {0, 1}}})`		- между точками (0, 0) и (0, 1). Этот узор совпадает с узором, который получается, если провести отрезок: - между (0, 1) и (0, 0) узор совпадает с узором (повернутым на 0 градусов), который был возвращен вторым запуском `BuildPattern`. Поэтому необходимо вернуть 1.

`GetCardNumber(`
`{{{1, 1}, {2, 2}},`
`{{1, 2}, {2, 0}}})`
	`3`	Передан узор из двух отрезков:
		- между $(1, 1)$ и $(2, 2)$,
		- между $(1, 2)$ и $(2, 0)$.
		Этот узор совпадает с узором, возвращенным первым запуском `BuildPattern`, повернутым на 90 градусов против часовой стрелки. Поэтому необходимо вернуть 3.

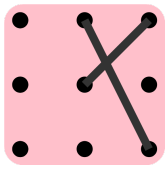
Второй запуск завершается.	-	-

Следующие три изображения задают:

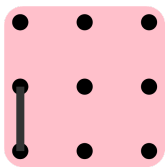
- Узор, который возвращает первый вызов `BuildPattern`:



- Узор, который передается как аргумент второму вызову `GetCardNumber`, он представляет собой узор выше, повернутый на 90 градусов против часовой стрелки.



- Узор, который возвращает второй вызов `BuildPattern`, он выглядит также как и узор, переданный как аргумент первому вызову `GetCardNumber`.



Sample Grader

Пример грейдера `grader.cpp` в архиве `Cheat.zip` из тестирующей системы считывает число q со стандартного потока ввода и затем q раз выполняет следующие действия:

- Считывает n из стандартного потока ввода
- Вызывает `BuildPattern(n)` и сохраняет возвращенное значение в переменной p.
- Вызывает `GetCardNumber(p)` и выводит возвращенное значение в стандартный поток вывода.

Если вы хотите внести изменения в грейдер, это можно сделать локально на вашем компьютере.

Чтобы скомпилировать пример грейдера с вашим решением, можно использовать следующую команду:

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

где `solution.cpp` — это ваше решение, которое вы планируете отправить в тестирующую систему. Чтобы запустить программу на примере из архива, можно выполнить следующую команду:

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

Обратите внимание, что в отличие от примера грейдера, грейдер в тестирующей системе будет вызывать функции в отдельных запусках вашей программы.