

Chika quiere hacer trampa

Nombre del problema	Cheat
Archivo de entrada	Interactive task
Archivo de salida	Interactive task
Límite de tiempo	2 seconds
Límite de memoria	512 megabytes

Chika tiene una baraja de q cartas numeradas con enteros positivos. Ella quiere jugar algunos juegos con sus amigos del consejo de estudiantes de la Academia Shuchi'in usando su baraja. Ella también quiere ganar, por lo que decide marcar secretamente el reverso de sus cartas.

Las cartas son cuadrados 2×2 donde la esquina inferior izquierda es $(0, 0)$ y la superior derecha es $(2, 2)$. Chika dibuja un cierto patrón en el reverso de cada carta para poder saber después, al mirar el patrón, qué número está escrito en el frente de la carta. Ella dibuja el patrón utilizando el siguiente procedimiento: las veces que ella quiera (posiblemente 0), toma dos puntos distintos A y B con coordenadas enteras, relativas a la esquina inferior izquierda de la carta, y traza un **segmento de línea recta** entre ellos.

Chika solo dibujará segmentos **válidos**; esto es, segmentos entre dos puntos A y B para los cuales no exista otro punto C (distinto de A y B) con coordenadas enteras que también se encuentre en el segmento. Por ejemplo, el segmento entre $(0, 0)$ y $(2, 2)$ es **inválido** dado que contiene el punto $(1, 1)$; en cambio, el segmento entre $(0, 0)$ y $(1, 1)$ y el segmento entre $(1, 1)$ y $(2, 2)$ son ambos **válidos**, y Chika puede incluso dibujar ambos segmentos en el mismo patrón. Además, nota que el segmento no es dirigido: un segmento dibujado de A a B es **idéntico** a sí mismo y al segmento dibujado en la dirección contraria, de B a A .

Chika quiere asegurarse que podrá reconocer las cartas sin importar cómo están rotadas. Una carta puede ser rotada $0, 90, 180$ o 270 grados en contra de las manecillas del reloj con respecto a la orientación original.

Tu tarea es ayudar a Chika a crear los patrones para las q cartas de su baraja y luego reconocer esas cartas más adelante.

Implementación

Este es un problema interactivo con dos etapas, **en donde cada etapa involucra una ejecución separada de tu programa**. Tendrás que implementar dos funciones:

- Una función `BuildPattern` que retorne el patrón que se va a dibujar en el reverso de la carta dada. Esta función será llamada q veces en la primera etapa.
- Una función `GetCardNumber` que retorne el número de una carta (posiblemente rotada) que contiene un patrón dado, dibujado durante la primera etapa. Esta función será llamada q veces en la segunda etapa.

La primera función,

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

toma un solo parámetro n , el número que está escrito en el frente de la carta. Debes retornar un `std::vector` que contenga los segmentos que Chika dibuja como patrón en el reverso de la carta para reconocerla después. Un segmento se representa como un `std::pair` de puntos, y un punto se representa como un `std::pair` (x, y) con coordenadas enteras y relativas a la esquina inferior izquierda de la carta, donde $0 \leq x, y \leq 2$. Todos los segmentos que dibuja Chika necesitan ser válidos y distintos entre sí. Se garantiza que todas las q llamadas a `BuildPattern` reciban diferentes valores para el parámetro n .

Después de recibir todos los patrones para las q cartas, el evaluador podrá hacer cualquiera de las siguientes operaciones, **un número de veces arbitrario**, sobre cada uno de los patrones:

- Rotar todo el patrón 0, 90, 180 o 270 en contra de las manecillas del reloj.
- Modificar el orden de los segmentos en el `std::vector` que representa el patrón.
- Cambiar el orden de los extremos de un segmento en el patrón (Un segmento dibujado de A a B puede transformarse al segmento idéntico que va de B a A).

La segunda función

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

toma un solo parámetro p , un `std::vector` de segmentos que describen el patrón dibujado por Chika en el reverso de la carta, obtenido de una llamada anterior a tu función `BuildPattern`. La función debe regresar el número n que está escrito en el frente de la carta. Recuerda que el patrón p no está necesariamente en la forma original que regresó la función `BuildPattern`; pudo haber sido objeto de las tres operaciones mencionadas anteriormente. También es posible que el orden de las cartas sea diferente del orden en el que fueron dadas en la primera etapa, pero se garantiza que cada carta será usada exactamente una vez.

Límites

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ para todas las llamadas a la función `BuildPattern`.
- Nota que existen algoritmos para construir patrones para distinguir al menos 67 000 000 cartas.

Subtareas

- Subtarea 1 (2 puntos): $n \leq 2$.
- Subtarea 2 (9 puntos): $n \leq 25$.
- Subtarea 3 (15 puntos): $n \leq 1\,000$ y el evaluador **no rotará** (operación de rotación) los patrones entre las etapas 1 y 2. El evaluador **podrá** realizar las otras dos operaciones.
- Subtarea 4 (3 puntos): $n \leq 16\,000\,000$ y el evaluador **no rotará** (operación de rotación) los patrones entre las etapas 1 y 2. El evaluador **podrá** realizar las otras dos operaciones.
- Subtarea 5 (24 puntos): $n \leq 16\,000\,000$.
- Subtarea 6 (18 puntos): $n \leq 40\,000\,000$.
- Subtarea 7 (29 puntos): Sin consideraciones adicionales.

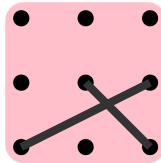
Interacción de ejemplo

Llamada a función	Valor retornado	Explicación
Empieza la primera ejecución.	-	-
<code>BuildPattern(3)</code>	<code>{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}</code>	Tenemos que crear un patrón para el número 3 en la carta de 2×2 . Decidimos dibujar 2 segmentos: - entre (0,0) y (2,1), - entre (1,1) y (2,0).
<code>BuildPattern(1)</code>	<code>{{{0, 1}, {0, 0}}}</code>	Tenemos que crear un patrón para el número 1 en la carta de 2×2 . Decidimos dibujar 1 segmento: - entre (0,1) y (0,0).
Termina la primera ejecución.	-	-
Empieza la segunda ejecución.	-	-
<code>GetCardNumber({{{0, 0}, {0, 1}}})</code>	1	Se nos da un patrón hecho de 1 segmento: - entre (0,0) y (0,1). Este es el mismo patrón que obtendríamos de dibujar el segmento:

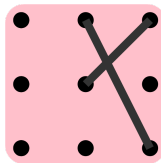
		- entre (0, 1) y (0, 0) el cual es exactamente el mismo patrón con la misma orientación (rotada 0 grados) que retornamos para la segunda llamada a la función <code>BuildPattern</code> . Por lo tanto, regresamos 1.
<code>GetCardNumber(</code> <code>{{{1, 1}, {2, 2}},</code> <code>{{1, 2}, {2, 0}}})</code>	3	Se nos da un patrón hecho de 2 segmentos: - entre (1, 1) y (2, 2), - entre (1, 2) y (2, 0). Este es el patrón que retorna en la primera llamada a la función <code>BuildPattern</code> , rotada 90 grados en contra de las manecillas del reloj. Por lo tanto, regresamos 3.
Termina la segunda ejecución.	-	-

Las siguientes tres imágenes representan, en ese orden:

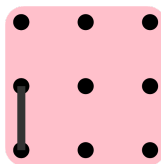
- El patrón regresado en la primera llamada a `BuildPattern`:



- El patrón recibido como parámetro en la segunda llamada a `GetCardNumber`, que es el primer patrón luego de ser rotado 90 grados en contra de las manecillas del reloj.



- El patrón regresado en la segunda llamada a `BuildPattern`, el cual es también el mismo patrón que se recibe como argumento en la primera llamada a `GetCardNumber`.



Evaluador de ejemplo

El evaluador de ejemplo provisto, `grader.cpp`, en el archivo adjunto `Cheat.zip` lee un entero q de la entrada estándar y luego realiza los siguientes pasos q veces:

- Lee un entero n de la entrada estándar.
- Llama a `BuildPattern(n)` y guarda el valor regresado en una variable p .
- Llama a `GetCardNumber(p)` e imprime el valor regresado en la salida estándar.

Puedes modificar el evaluador localmente si así lo deseas.

Para compilar el evaluador de ejemplo con tu solución, puedes usar el siguiente comando en la consola:

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

donde `solution.cpp` es tu archivo de solución que enviarás a CMS. Para ejecutar el programa con la entrada del archivo, copia el siguiente comando en la consola:

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

Por favor nota que, a diferencia del evaluador de ejemplo, el verdadero evaluador del CMS realizará la primera etapa y la segunda etapa en ejecuciones separadas de tu programa.