Chika Wants to Cheat

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

Chika har en kortlek med q kort numrerade med diverse positiva heltal. Hon vill spela några spel med sina vänner från elevrådet med dessa kort, men hon vill också vinna, så hon bestämmer sig för att i hemlighet markera baksidan av korten i sin kortlek.

Alla kort är formade som kvadrater av storleken 2×2, där det nedre vänstra hörnet har koordinater $(0, 0)$ och det övre högra hörnet har koordinater $(2, 2)$. Chika ritar ett speciellt mönster på baksidan av varje kort så att hon senare kan veta vilket nummer som står på framsidan av kortet genom att titta på mönstret. Hon ritar ett sådant mönster med följande process: Så många gånger hon vill (möjligtvis 0 gånger) väljer hon två olika punkter A och B som har heltalskoordinater relativt det nedre vänstra hörnet av kortet och ritar ett **rakt linjesegment** mellan dem.

Chika vill bara rita **giltiga** segment, vilket betyder att det inte får finnas någon annan punkt C mellan A och B (och skild från A och B) med heltalskoordinat som också ligger på segmentet. Till exempel, segmentet mellan $(0, 0)$ och $(2, 2)$ är **inte giltigt** eftersom det innehåller punkten $(1, 1)$, men segmenten mellan $(0, 0)$ och $(1, 1)$ och mellan $(1, 1)$ och $(2, 2)$ är båda **giltiga**, och Chika kan till och med rita båda på samma mönster. Notera också att segmenten har ingen riktning: Ett segment som ritas från A till B är **identiskt** till både sig själv och segmentet som ritas i motsatt riktning, från B till A.

Speciellt vill Chika se till att hon kan känna igen sina kort oavsett hur de är roterade. Ett kort kan roteras 0, 90, 180 eller 270 grader moturs relativt den ursprungliga orienteringen.

Din uppgift är att hjälpa Chika designa mönstren för de q korten kortleken och sedan känna igen korten.

Implementation

Detta är ett interaktivt problem med två faser, ***varje fas innebär en separat körning av ditt program***. Du behöver implementera två funktioner:

- En `BuildPattern`-funktion som returnerar mönstret som ska ritas på baksidan av ett givet kort. Denna funktion kommer anropas q gånger i den första fasen.
- En `GetCardNumber`-funktion som returnerar numret av ett (möjligtvis roterat) kort som har ett givet mönster ritat i den första fasen. Denna funktion kommer anropas q gånger i den andra fasen.

Den första funktionen

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

tar en parameter n, numret som står på framsidan av kortet. Du behöver returnera en `std::vector` med segmenten som Chika ritar som ett mönster på baksidan av kortet för att kunna urskilja det senare. Ett segment representeras som ett `std::pair` av punkter, och en punkt representeras som ett `std::pair` (x, y) av heltalskoordinater relativt den lägre vänstra hörnet av kortet, där $0 \leq x, y \leq 2$. Alla segment som Chika ritar måste vara giltiga och parvis icke-identiska. Det är garanterat att alla q anrop till BuildPattern anropas med olika värden för parametern n.

Efter att ha tagit emot alla mönster för de q korten, kan gradern göra någon av följande operationer upprepade antal gånger på varje mönster:

- Rotera hela mönstret med 0, 90, 180 eller 270 grader moturs.
- Ändra ordning på segmenten i `std::vector`-representationen av mönstret.
- Ändra ordning på ändpunkterna av ett segment i mönstret (ett segment som ritades från A till B skulle kunna bli det identiska segmentet från B till A).

Den andra funktionen,

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

tar en parameter p, en `std::vector` av segment som beskriver mönstret som ritades av Chika på baksidan av ett kort, baserat på returvärdet av ett tidigare anrop till din `BuildPattern`-funktion. Funktionen måste returnera numret n som står på framsidan av kortet. Kom ihåg att mönstret p inte nödvändigtvis är i den ursprungliga formen som returnerades av `BuildPattern` - det kan vara så att den har genomgått de tre operationerna som beskrevs ovan. Det är också möjligt att ordningen på korten är annorlunda från ordningen de gavs i den förta fasen.

Begränsningar

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ för alla anrop till `BuildPattern`.
- Notera att det finns algoritmer för att konstruera mönster så att det går att särskilja $67\,000\,000$ olika kort.

Poängsättning

- Subtask 1 (2 poäng): $n \leq 2$.
- Subtask 2 (9 poäng): $n \leq 25$.
- Subtask 3 (15 poäng): $n \leq 1\,000$ och gradern kommer **inte rotera** mönstren mellan fas 1 och 2. (gradern **kan** utföra de andra två operationerna.)
- Subtask 4 (3 poäng): $n \leq 16\,000\,000$ och gradern kommer **inte rotera** mönstren mellan fas 1 och 2. (gradern **kan** utföra de andra två operationerna.)
- Subtask 5 (24 poäng): $n \leq 16\,000\,000$.
- Subtask 6 (18 poäng): $n \leq 40\,000\,000$.
- Subtask 7 (29 poäng): Inga ytterliga begränsningar.

Exempelinteraktion

Funktionsanrop	Returvärde	Förklaring
Första fasen börjar.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Vi behöver skapa ett mönster för numret 3 på 2×2-kortet. Vi ritar 2 segment: - mellan $(0, 0)$ och $(2, 1)$, - mellan $(1, 1)$ och $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Vi behöver skapa ett mönster för numret 1 på 2×2-kortet. Vi ritar 1 segment: - mellan $(0, 1)$ och $(0, 0)$.
Första fasen slutar.	-	-
Andra fasen börjar.	-	-
`GetCardNumber( {{{0, 0}, {0, 1}}})`	`1`	Vi får ett mönster bestående av 1 segment: - mellan $(0, 0)$ och $(0, 1)$. Detta är samma mönster vi skulle få av att rita segmentet: - mellan $(0, 1)$ och $(0, 0)$. vilket är exakt samma mönster med samma orientering (roterat 0 grader) som vi returnerade i det andra anropet

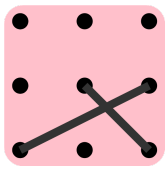
		till funktionen `BuildPattern`. Därför måste vi returnera 1.

`GetCardNumber(`
`{{{1, 1}, {2, 2}},`
`{{1, 2}, {2, 0}}})`
	`3`	Vi får ett mönster bestående av 2 segment:
		- mellan $(1, 1)$ och $(2, 2)$,
		- mellan $(1, 2)$ och $(2, 0)$.
		Detta är samma mönster som vi returnerade i det första anropet till `BuildPattern` roterat 90 grader moturs. Därmed returnerar vi 3.

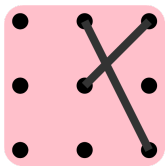
Andra fasen slutar.	-	-

De nästa tre bilderna representerar, i ordning:

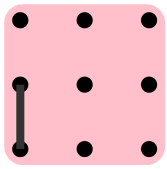
- Mönstret som returnerades av det första anropet till `BuildPattern`:



- Mönstret som skickades som parameter till det andra anropet av `GetCardNumber`, vilket är det första mönstret efter det roterats 90 grader moturs.



- Mönstret som returnerades av det andra anropet till `BuildPattern`, vilket är samma som mönstret som skickades som parameter till det förta anropet av `GetCardNumber`.



Sample Grader

Den givna sample-gradern, `grader.cpp`, i bilagan `Cheat.zip`, läser ett heltal q från standard input och gör sedan följande steg q gånger:

- Läs ett heltan n från standard input.
- Anropa `BuildPattern(n)` och lagra värdet i variabeln p.
- Anropa `GetCardNumber(p)` och skriv returvärdet till standard output.

Du kan ändra din grader lokalt om du vill.

För att kompilera sample-gradern med din lösning kan du använda följande kommando i terminalen:

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

där `solution.cpp` är din lösningsfil som du kommer skicka in till CMS. För att köra programmet med exempelindatan som finns i bilagan, skriv följande kommando i terminalen:

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

Notera att, till skillnad från sample-gradern, kommer den riktiga gradern på CMS att utföra de första och andra faserna i separata exekveringar av ditt program.