Chika Wants to Cheat

Aufgabenname	Cheat
Eingabedatei	Interaktive Aufgabe
Ausgabedatei	Interaktive Aufgabe
Zeitlimit	2 Sekunden
Speicherlimit	512 Megabytes

Chika hat ein Deck mit q Spielkarten, welche durch verschiedene positive ganze Zahlen gekennzeichnet sind. Sie möchte mit ihren Freunden aus dem Schülerrat der Shuchi'in Akademie Karten spielen. Sie möchte jedoch gewinnen, und markiert deswegen heimlich die Rückseiten ihrer Karten.

Die Karten sind quadratisch, mit der Grösse 2×2, wobei die linke, untere Ecke die Koordinaten $(0, 0)$ hat und die rechte, obere Ecke die Koordinaten $(2, 2)$. Chika zeichnet ein gewisses Muster auf die Rückseite jeder Karte, sodass sie später anhand des Musters auf die Zahl auf der Vorderseite der Karte schliessen kann. Sie zeichnet ein Muster wie folgt: So oft sie will (möglicherweise 0 mal), wählt sie zwei verschiedene Punkt A und B mit ganzzahligen Koordinaten relativ zur der unteren linken Ecke der Karte und zeichnet eine **gerade Linie** zwischen ihnen.

Chika wird jedoch nur **gültige** Linien zeichnen, definiert als Linien zwischen zwei Punkten A und B, so dass es keinen anderen Punkt C (verschieden von A und B) mit ganzzahligen Koordinaten gibt, der auch auf dieser Linie liegt. Zum Beispiel ist die Linie zwischen $(0, 0)$ und $(2, 2)$ **ungültig**, da sie den Punkt $(1, 1)$ enthält. Doch die Linien zwischen $(0, 0)$ und $(1, 1)$, sowie zwischen $(1, 1)$ und $(2, 2)$ sind beide **gültig** und Chika kann sogar beide auf das selbe Muster zeichnen. Beachte, dass Linien keine Richtung haben: die Linie zwischen A und B ist **identisch** zu der Linie zwischen B und A.

Wichtig ist auch, dass Chika ihre Karte erkennen können möchte, egal wie diese gedreht sind. Eine Karte kann in Bezug zu ihrer Ausgangsposition um 0, 90, 180 oder 270 Grad gegen den Uhrzeigersinn gedreht sein.

Deine Aufgabe besteht darin, Chika beim Design der Muster für die q Karten in ihrem Deck zu helfen und diese später wieder zu erkennen.

Implementierung

Dies ist eine interaktive Aufgabe mit zwei Phasen, ***wobei jede Phase eine separaten Ausführung deines Programms beinhaltet.***. Implementiere folgende zwei Funktionen:

- Eine Funktion `BuildPattern`, die das Muster, welches auf die Rückseite einer gegebene Karte gezeichnet werden soll, zurückgibt. Diese Funktion wird in der ersten Phase q mal aufgerufen.
- Eine Funktion `GetCardNumber`, welche die Nummer einer (möglicherweise gedrehten) Karte zurückgibt, die ein gegebenes, in der ersten Phase gezeichnetes, Muster aufweist. Diese Funktion wird in der zweiten Phase q mal aufgerufen.

Die erste Funktion

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

hat einen Parameter n - die Zahl, welche auf der Vorderseite der Karte steht. Du musst einen `std::vector` zurückgeben, welcher die Linien enthält, die Chika als Muster auf die Rückseite der Karte zeichnet, um sie später wiederzuerkennen. Eine Linie wird als ein `std::pair` von Punkten dargestellt, und ein Punkt wird dargestellt als ein `std::pair` (x, y) von ganzzahligen Koordinaten relativ zur unteren linken Ecke der Karte, wobei $0 \leq x, y \leq 2$. Alle von Chika gezeichneten Linien müssen gültig und paarweise verschieden sein. Es ist garantiert, dass alle q Aufrufe von `BuildPattern` verschiedene Werte für n erhalten.

Nachdem der Grader alle Muster für die q Karten erhalten hat, kann er für jedes Muster folgende Operationen beliebig oft ausführen:

- Drehe das ganze Muster um 0, 90, 180 oder 270 Grad gegen den Uhrzeigersinn.
- Verändere die Reihenfolge der Linien in der `std::vector` Darstellung des Musters.
- Verändere die Reihenfolge der Endpunkte einer Linie des Musters. (Eine Linie von A nach B kann durch die Linie von B nach A ersetzt werden).

Die zweite Funktion

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

hat einen einzigen Parameter p, ein `std::vector` mit Linien, welcher das Muster beschreibt, das Chika auf die Rückseite der Karte gezeichnet hat - basierend auf dem Rückgabewert eines vorherigen Aufrufs deiner `BuildPattern` Funktion. Die Funktion muss die Zahl n zurückgeben, welche sich auf der Vorderseite der Karte befindet. Beachte, dass das Muster p nicht notwendigerweise in der Originalform ist, wie `BuildPattern` sie zurückgegeben hat; die drei oben genannten Operationen könnten darauf angewendet worden sein. Es ist auch möglich, dass

die Reihenfolge der Karten anders ist als sie in der ersten Phase gegeben wurden, aber es ist garantiert, dass jede Karte genau einmal abgefragt wird.

Beschränkungen

- $1 \leq q \leq 10'000$.
- $1 \leq n \leq 67'000'000$ für alle Aufrufe der Funktion `BuildPattern`.
- Beachte, dass es Algorithmen gibt, um Muster so zu konstruieren, dass man $67'000'000$ verschiedene Karten erkennen kann.

Teilaufgaben

- Teilaufgabe 1 (2 Punkte): $n \leq 2$.
- Teilaufgabe 2 (9 Punkte): $n \leq 25$.
- Teilaufgabe 3 (15 Punkte): $n \leq 1'000$ und der Grader **dreht die Muster nicht** zwischen der ersten und der zweiten Phase. Der Grader **kann** jedoch die anderen beiden Operationen anwenden.
- Teilaufgabe 4 (3 Punkte): $n \leq 16'000'000$ und der Grader **dreht die Muster nicht** zwischen der ersten und der zweiten Phase. Der Grader **kann** jedoch die anderen beiden Operationen anwenden.
- Teilaufgabe 5 (24 Punkte): $n \leq 16'000'000$.
- Teilaufgabe 6 (18 Punkte): $n \leq 40'000'000$.
- Teilaufgabe 7 (29 Punkte): Keine weiteren Beschränkungen.

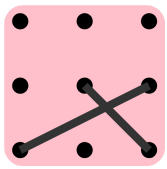
Beispielinteraktion

Funktionsaufruf	Rückgabewert	Erklärung
Beginn der ersten Ausführung.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Wir sollen ein Muster für die Zahl 3 auf der 2×2 grossen Karte zeichen. Wir entscheiden uns, 2 Linien zu zeichnen: - zwischen $(0, 0)$ und $(2, 1)$, - zwischen $(1, 1)$ und $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Wir sollen ein Muster für die Zahl 1 auf der 2×2 grossen Karte zeichen. Wir entscheiden uns, eine Linie zu zeichnen: - zwischen $(0, 1)$ und $(0, 0)$.
Ende der ersten Ausführung.	-	-
Beginn der zweiten	-	-

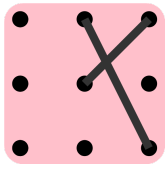
Ausführung.		
`GetCardNumber({{{0, 0}, {0, 1}}})`	`1`	Wir erhalten ein Muster, das aus einer Linie besteht: - zwischen $(0,0)$ und $(0,1)$. Dies ist das selbe Muster, das wir durch Zeichnen der Linie zwischen $(0,1)$ und $(0,0)$ erhalten würden, das genau das selbe Muster ist, mit der selben Orientierung (um 0 Grad gedreht), das wir beim zweiten Aufruf der Funktion `BuildPattern` zurückgegeben haben. Deshalb geben wir 1 zurück.
`GetCardNumber({{{1, 1}, {2, 2}}, {{1, 2}, {2, 0}}})`	`3`	Wir erhalten ein Muster, das aus 2 Linien besteht: - zwischen $(1,1)$ und $(2,2)$, - zwischen $(1,2)$ und $(2,0)$. Dies ist das Muster, das wir beim ersten Aufruf der `BuildPattern` Funktion zurückgegeben haben, nur um 90 Grad gegen den Uhrzeigersinn gedreht. Deshalb geben wir 3 zurück.
Ende der zweiten Ausführung.	-	-

Die nächsten drei Bilder stellen in dieser Reihenfolge dar:

- Das beim ersten Aufruf von `BuildPattern` zurückgegebene Muster:

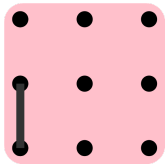


- Das beim zweiten Aufruf von `GetCardNumber` als Parameter übergebenes Muster, welches auch das erste erzeugte Muster nach einer Drehung um 90 Grad gegen den Uhrzeigersinn ist:



- Das beim zweiten Aufruf von `BuildPattern` zurückgegebene Muster, welches auch das selbe Muster ist, was beim ersten Aufruf von `GetCardNumber` als Parameter übergeben

wurde.



Beispielgrader

Der zur Verfügung gestellte Beispielgrader, `grader.cpp`, im Anhang der Aufgabe, `Cheat.zip`, liest eine ganze Zahl q von der Standardeingabe und wird dann die folgenden Schritte q mal ausführen:

- Liest eine ganze Zahl n von der Standardausgabe.
- Ruft `BuildPattern(n)` auf und speichert den Rückgabewert in der Variable p.
- Ruft `GetCardNumber(p)` auf und gibt den Rückgabewert auf der Standardausgabe aus.

Du kannst bei Bedarf deinen Beispielgrader lokal modifizieren.

Um den Beispielgrader mit deiner Lösung zu kompilieren, kannst du den folgenden Befehl auf deinem Terminal benutzen:

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

wobei `solution.cpp` deine Lösungsdatei ist, welche du an das CMS übermittelst. Um das Programm mit der Beispieleingabe aus der Aufgabenstellung (welche im Anhang zu finden ist) auszuführen, tippe den folgenden Befehl in dein Terminal:

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

Beachte, dass der echte Grader auf dem CMS im Gegensatz zum Beispielgrader die erste und zweite Phase in separaten Aufrufen deines Programmes ausführt.