Jadzia gra w kenta

Nazwa zadania	Kent
Wejście	Zadanie interaktywne
Wyjście	Zadanie interaktywne
Limit czasu	2 sekundy
Limit pamięci	512 MB

Jadzia ma talię q kart ponumerowanych liczbami całkowitymi. Chce pograć w kenta ze swoimi znajomymi z Ukrainy, ale lubi wygrywać, więc zdecydowała się potajemnie oznaczyć tył kart w swojej talii.

Karty mają kształt kwadratu o wymiarach 2×2. Lewy dolny róg karty ma współrzędne $(0, 0)$, a prawy górny róg ma współrzędne $(2, 2)$. Jadzia rysuje pewien wzór z tyłu każdej karty, by później, po zerknięciu na symbol, wiedzieć, jaka liczba znajduje się z przodu karty. Rysuje wzór, używając następującej procedury: tak wiele razy, jak chce (możliwe, że 0 razy), wybiera dwa różne punkty A i B o współrzędnych całkowitych względem lewego dolnego rogu karty i łączy je **odcinkiem**.

Jadzia rysuje wyłącznie **odpowiednie** odcinki, czyli odcinki pomiędzy dwoma punktami A i B, dla których nie istnieje punkt C (różny od A i B) o współrzędnych całkowitych, który też leży na tym odcinku. Przykładowo, odcinek pomiędzy $(0, 0)$ i $(2, 2)$ jest nieodpowiedni, ponieważ zawiera punkt $(1, 1)$. Odcinki łączące $(0, 0)$ i $(1, 1)$ oraz łączące $(1, 1)$ i $(2, 2)$ są odpowiednie i Jadzia może je narysować w tym samym wzorze. Zauważ, że odcinki nie mają kierunku, a więc odcinek narysowany z A do B jest **identyczny** z samym sobą i odcinkiem narysowanym w przeciwnym kierunku, to jest z B do A.

Jadzia chce się upewnić, że rozpozna swoje karty bez względu na to, jak będą obrócone. Karta może być obrócona o 0, 90, 180 lub 270 stopni przeciwnie do ruchu wskazówek zegara zaczynając od początkowego ustawienia.

Twoim zadaniem jest pomóc Jadzi zaprojektować wzory dla q kart w jej talii, a potem rozpoznać razem z nią te karty.

Implementacja

*To jest interaktywne zadanie dwuetapowe, **a oba etapy wymagają oddzielnych uruchomień Twojego programu***. Powinnaś zaimplementować dwie funkcje:

- Funkcję `BuildPattern` zwracającą wzór, który powinien być narysowany z tyłu danej karty. Ta funkcja zostanie wywołana q razy w pierwszym etapie.
- Funkcję `GetCardNumber` zwracającą liczbę na karcie, na której Jadzia narysowała dany wzór w pierwszym etapie. Karta może być obrócona. Ta funkcja zostanie wywołana q razy w drugim etapie.

Pierwsza funkcja

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

przyjmuje pojedynczy argument n, czyli liczbę, która znajduje się z przodu karty. Powinna zwrócić `std::vector` zawierający odcinki, które Jadzia rysuje jako wzór z tyłu tej karty, aby ją później rozpoznać. Odcinek jest reprezentowany jako para (`std::pair`) punktów, natomiast punkt jako para (`std::pair`) liczb całkowitych (x, y) będących współrzędnymi względem lewego dolnego rogu karty, gdzie $0 \leq x, y \leq 2$. Wszystkie odcinki, które narysuje Jadzia, muszą być odpowiednie i parami różne. Wszystkie zapytania do funkcji BuildPattern podadzą inną wartość n.

Po otrzymaniu wzorów dla wszystkich q kart, sprawdzaczka może wykonywać następujące operacje dowolną liczbę razy dla każdego ze wzorów:

- Obrócić cały wzór o 0, 90, 180 lub 270 stopni przeciwnie do ruchu wskazówek zegara.
- Zmodyfikować kolejność odcinków w wektorowej (`std::vector`) reprezentacji wzoru.
- Zmienić kolejność końców odcinka we wzorze (odcinek łączący A i B może stać się identycznym odcinkiem łączącym B i A).

Druga funkcja

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

przyjmuje pojedynczy argument p będący wektorem (`std::vector`) odcinków opisujących wzór, który został narysowany przez Jadzię - na podstawie zwróconej wcześniej wartości przy wywołaniu funkcji `BuildPattern`. Funkcja powinna zwrócić liczbę n znajdującą się z przodu karty. Zauważ, że wzór p niekoniecznie jest w tej samej formie, która jest zwracana przez `BuildPattern`; owy wzór mógł być zmieniany przez trzy operacje wspomniane wyżej. Jest możliwe, że sprawdzaczka wywoła powyższą funkcję pytając o karty w kolejności innej, niż początkowa, ale nie zapyta o tę samą kartę wielokrotnie.

Ograniczenia

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ dla każdego wywołania funkcji `BuildPattern`.
- Można udowodnić, że istnieją algorytmy konstruujące wzory, które pozwolą na rozpoznawanie $67\,000\,000$ różnych kart.

Podzadania

- Podzadanie 1 (2 punkty): $n \leq 2$.
- Podzadanie 2 (9 punktów): $n \leq 25$.
- Podzadanie 3 (15 punktów): $n \leq 1\,000$ i sprawdzaczka **nie będzie obracać** wzorów pomiędzy pierwszym a drugim etapem (sprawdzaczka **może** wykonywać dwie pozostałe operacje).
- Podzadanie 4 (3 punkty): $n \leq 16\,000\,000$ i sprawdzaczka **nie będzie obracać** wzorów pomiędzy pierwszym a drugim etapem (sprawdzaczka **może** wykonywać dwie pozostałe operacje).
- Podzadanie 5 (24 punkty): $n \leq 16\,000\,000$.
- Podzadanie 6 (18 punktów): $n \leq 40\,000\,000$.
- Podzadanie 7 (29 punktów): Brak dodatkowych ograniczeń.

Przykład

Wywołanie funkcji	Zwrócona wartość	Wyjaśnienie
Zaczyna się pierwszy etap.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Musimy stworzyć wzór dla numeru 3 na karcie o rozmiarze 2×2. Decydujemy, że narysujemy 2 odcinki: - łączący $(0,0)$ i $(2,1)$, - łączący $(1,1)$ i $(2,0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Musimy stworzyć wzór dla liczby 1 na karcie o rozmiarze 2×2. Decydujemy, że narysujemy 1 odcinek: - łączący $(0,1)$ i $(0,0)$.
Kończy się pierwszy etap.	-	-
Zaczyna się drugi etap.	-	-
`GetCardNumber( {{{0, 0}, {0, 1}}})`	`1`	Dostajemy wzór stworzony z 1 odcinka: - łączącego $(0,0)$ i $(0,1)$. Jest on dokładnie taki sam, jak wzór otrzymany poprzez narysowanie odcinka: - łączącego $(0,1)$ i $(0,0)$

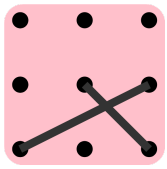
		Wzór jest identyczny (nawet pod względem orientacji, gdyż jest obrócony o 0 stopni) ze wzorem otrzymanym przy drugim wywołaniu funkcji `BuildPattern`. Wobec tego zwracamy 1.

`GetCardNumber(`
`{{{1, 1}, {2, 2}},`
`{{1, 2}, {2, 0}}})`
	`3`	Dostajemy wzór stworzony z 2 odcinków:
		- łączącego $(1, 1)$ i $(2, 2)$,
		- łączącego $(1, 2)$ i $(2, 0)$.
		Jest to wzór, który otrzymaliśmy przy pierwszym wywołaniu funkcji `BuildPattern` obrócony o 90 stopni przeciwnie do ruchu wskazówek zegara. Wobec tego zwracamy 3.

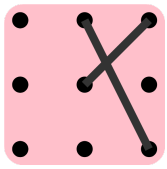
Kończy się drugi etap.	-	-

Następne trzy obrazki ukazują odpowiednio:

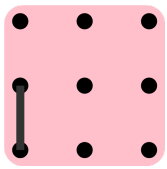
- Wzór zwrócony jako wynik po pierwszym wywołaniu `BuildPattern`:



- Wzór otrzymany jako argument przy drugim wywołaniu `GetCardNumber`, który jest pierwszym wzorem obróconym o 90 stopni przeciwnie do ruchu wskazówek zegara.



- Wzór zwrócony jako wynik po drugim wywołaniu `BuildPattern`, który jest zarazem wzorem otrzymanym jako argument przy pierwszym wywołaniu `GetCardNumber`.



Sprawdzaczka pomocnicza

Sprawdzaczka pomocnicza, `grader.cpp`, znajdująca się w załączniku `Cheat.zip`, wczytuje liczbę całkowitą q ze standardowego wejścia, a potem wykonuje następujące kroki q razy:

- Wczytuje liczbę n ze standardowego wejścia.
- Wywołuje `BuildPattern(n)` i przechowuje zwróconą wartość w zmiennej p.
- Wywołuje `GetCardNumber(p)` i wypisuje zwróconą wartość na standardowe wyjście.

Możesz lokalnie modyfikować sprawdzaczkę, jeśli uznasz to za stosowne.

Aby skompilować sprawdzaczkę pomocniczą ze swoim rozwiązaniem, możesz użyć następującej komendy w terminalu:

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

gdzie `solution.cpp` jest plikiem z rozwiązaniem, który chcesz wysłać do CMS. Aby uruchomić program z przykładowym wejściem z załącznika, wpisz następującą komendę w terminalu:

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

W przeciwieństwie do sprawdzaczki pomocniczej, prawdziwa sprawdzaczka w CMS będzie wykonywać pierwszy etap i drugi etap w osobnych uruchomieniach Twojego programu.