Chika wil valsspelen

Probleemnaam	Cheat
Invoerbestand	Interactive task
Uitvoerbestand	Interactive task
Tijdslimiet	2 seconden
Geheugenlimiet	512 megabytes

Chika heeft een kaartendeck van q speelkaarten genummerd met verscheidene positieve integers. Ze wil een aantal spellen spelen met haar vrienden van de Shuchi'in Academy leerlingenraad met deze kaarten. Ze wil graag winnen, daarom besluit ze stiekem de achterzijde van haar kaarten te markeren.

De kaarten zijn allemaal vierkant en hebben de grootte 2×2, waarbij de hoek linksonder de coördinaten $(0, 0)$ heeft en rechtsboven de coördinaten $(2, 2)$. Chika tekent een bepaald patroon op de achterzijde van elke kaart, zodat ze later weet, bij het bekijken van het patroon, welk nummer op de voorzijde van de kaart staat. Ze tekent een patroon volgens de volgende procedure: zo vaak als ze wil (mogelijk 0 keer), kiest ze twee unieke punten, A en B, die integer coördinaten hebben relatief ten opzichte van de linksonderhoek van de kaart en ze tekent een **rechte lijnsegment** tussen de punten.

Chika tekent enkel **geldige** segmenten, wat betekent dat voor segmenten tussen de twee punten A en B er geen ander punt C (anders dan A en B) met integer coördinaten is dat ook op het segment ligt. Bijvoorbeeld, het segment tussen $(0, 0)$ en $(2, 2)$ is **ongeldig** omdat het ook punt $(1, 1)$ bevat, maar de segmenten tussen $(0, 0)$ en $(1, 1)$ en tussen $(1, 1)$ en $(2, 2)$ zijn beiden **geldig**, en Chika kan zelfs beiden gebruiken in hetzelfde patroon. Houd er rekening mee dat de segmenten geen richting hebben; een segment getekend van A naar B is **identiek** aan zichzelf en aan het segment in de omgekeerde richting, van B naar A.

Belangrijk: Chika wil zeker weten dat ze haar kaarten herkend ongeacht hoe ze zijn gedraaid. Een kaart kan 0, 90, 180 of 270 graden tegen de klok in gedraaid worden ten opzichte van de beginpositie.

Het is jouw taak om Chika te helpen om patronen te ontwerpen voor q kaarten en deze kaarten later te kunnen herkennen.

Implementatie

Dit is een interactieve taak met twee fases, ***voor elke fase wordt je programma los uitgevoerd****.* Je moet de volgende twee functies implementeren:

- Een `BuildPattern` functie welke een patroon terug geeft dat getekend moet worden op de achterkant van de meegegeven kaart. Deze functie wordt q keer aangeroepen in de eerste fase.
- Een `GetCardNumber` functie dat het cijfer terug geeft van een (mogelijke gedraaide) kaart dat het meegegeven patroon bevat wat is getekend in de eerste fase. Deze functie wordt q keer aangeroepen in de tweede fase.

De eerste functie,

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

ontvangt een enkele parameter n, het cijfer dat is genoteerd op de voorzijde van de kaart. Je moet een `std::vector` teruggeven welke de segmenten bevat die Chika als patroon op de achterzijde van de kaart tekent om deze later te kunnen herkennen. Een segment wordt gerepresenteerd als een `std::pair` van punten en een punt wordt gerepresenteerd als een `std::pair` (x, y) van integer coördinaten relatief ten opzichte van de linkeronderhoek van de kaart, waarbij $0 \leq x, y \leq 2$. Alle segmenten die Chika tekent moeten geldig zijn en paarsgewijs niet-identiek. Het is gegarandeerd dat all q aanroepen naar `BuildPattern` verschillende waardes bevatten voor de parameter n.

Na het ontvangen van alle patronen voor de q kaarten, kan de grader elk van de volgende bewerkingen uitvoeren, een willekeurig aantal keren, op elk patroon:

- Draai het volledige patroon tegen de klok in met 0, 90, 180 of 270 graden.
- De volgorde van de segmenten wijzigen in de `std::vector` die het patroon representeert.
- De volgorde wijzigen van de punten van een segment in het patroon. (Een segment getekend van A naar B kan een identiek segment worden van B naar A.)

De tweede functie,

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

ontvangt een enkele parameter p, een `std::vector` van segmenten die het patroon beschrijven dat door Chika is getekend op de achterzijde van de kaart, gebaseerd op de ontvangen waarde van de eerdere aanroep naar jouw `BuildPattern` functie. De functie moet het cijfer n terug geven dat is genoteerd op de voorzijde van de kaart. Let op: het patroon p is niet per se in het originele formaat zoals teruggegeven door `BuildPattern`; het kan onderworpen zijn aan de bewerkingen

zoals hierboven vermeld. Het is ook mogelijk dat de volgorde van de kaarten anders is dan ingevoerd in de eerste fase, maar het is gegarandeerd dat iedere kaart exact één keer gebruikt wordt.

Randvoorwaarden

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ voor alle aanroepen naar de functie `BuildPattern`.
- Let op: er bestaan algoritmes om patronen te construeren zodat het mogelijk is om $67\,000\,000$ verschillende kaarten te kunnen herkennen.

Puntentelling

- Subtaak 1 (2 punten): $n \leq 2$.
- Subtaak 2 (9 punten): $n \leq 25$.
- Subtaak 3 (15 punten): $n \leq 1\,000$ en de grader zal de patronen **niet draaien** tussen de fases 1 en 2. (De grader **kan** wel de andere twee bewerkingen uitvoeren.)
- Subtaak 4 (3 punten): $n \leq 16\,000\,000$ en de grader zal de patronen **niet draaien** tussen de fases 1 en 2. (De grader **kan** wel de andere twee bewerkingen uitvoeren.)
- Subtaak 5 (24 punten): $n \leq 16\,000\,000$.
- Subtaak 6 (18 punten): $n \leq 40\,000\,000$.
- Subtaak 7 (29 punten): Geen aanvullende randvoorwaarden.

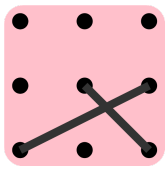
Interactie voorbeeld

Functieaanroep	Returnwaarde	Toelichting
Eerste fase begint.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	We moeten een patroon maken voor het cijfer 3 op de 2×2 kaart. We besluiten om 2 segmenten te tekenen: - tussen $(0, 0)$ en $(2, 1)$, - tussen $(1, 1)$ en $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	We moeten een patroon maken voor het cijfers 1 op de 2×2 kaart. We besluiten om 1 segment te tekenen: - tussen $(0, 1)$ en $(0, 0)$.
Eerste fase eindigt.	-	-
Tweede fase begint.	-	-
`GetCardNumber({{{0, 0}, {0, 1}}})`	`1`	We ontvangen een patroon bestaande uit 1 segment: - tussen $(0, 0)$ en $(0, 1)$.

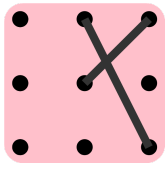
		Dit is hetzelfde patroon als wanneer we het volgende segment zouden tekenen: - tussen $(0, 1)$ en $(0, 0)$ wat exact hetzelfde patroon is met exact dezelfde positie (0 graden gedraaid) als het patroon dat we hebben ontvangen bij de tweede aanroep naar de functie `BuildPattern`. Daarom geven we 1 terug.
`GetCardNumber({{{1, 1}, {2, 2}}, {{1, 2}, {2, 0}}})`	`3`	We ontvangen een patroon bestaande uit 2 segmenten: - tussen $(1, 1)$ en $(2, 2)$, - tussen $(1, 2)$ en $(2, 0)$. Dit is het patroon dat we hebben ontvangen bij de eerste aanroep naar de functie `BuildPattern`, 90 graden tegen de klok in gedraaid. Daarom geven we 3 terug.
Tweede fase eindigt.	-	-

De volgende drie afbeeldingen tonen:

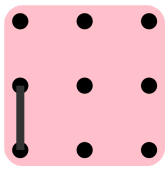
- Het patroon zoals teruggegeven na de eerste aanroep van `BuildPattern`:



- Het patroon zoals meegegeven als parameter bij de tweede aanroep van `GetCardNumber`, wat het eerste patroon is wat 90 graden tegen de klok in is gedraaid.



- Het patroon zoals teruggegeven na de tweede aanroep van `BuildPattern`, wat hetzelfde patroon is zoals meegegeven bij de eerste aanroep van `GetCardNumber`.



Voorbeeld grader

De gegeven voorbeeld grader, `grader.cpp`, in de bijlage van de taak `Cheat.zip`, leest een integer q van de standaard invoer en zal dan vervolgens de volgende stappen q keer uitvoeren:

- Lees een integer n van standaard invoer.
- Roep `BuildPattern(n)` aan en sla de terug gegeven waarde op als variabele p.
- Roep `GetCardNumber(p)` aan en print de terug gegeven waarde naar de standaard uitvoer.

Je kan de grader lokaal bewerken indien gewenst.

Om de sample grader te compileren met jouw oplossing, kun je het volgende commando gebruiken in de terminal prompt:

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

waarin `solution.cpp` het bestand is waarin jouw oplossing staat die je wil submitten in CMS. Om het programma uit te voeren met de voorbeeldinvoer uit de bijlage, kun je het volgende commando gebruiken in de terminal prompt:

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

Let op, in tegenstelling tot de voorbeeld grader, voert de daadwerkelijke grader op CMS de eerste en tweede fase los uit.