Chika Wants to Cheat

Oppgavenavn	Cheat
Inputfil	Interaktiv oppgave
Outputfil	Interaktiv oppgave
Tidsbegrensning	2 sekunder
Minnebegrensning	512 megabytes

Chika har en kortstokk med q kort nummerert med forskjellige positive heltall. Hun ønsker å spille kort med vennene hennes fra Shuchi'in akademiet ved hjelp av disse kortene, men hun ønsker også å vinne, og bestemmer seg derfor for å i hemmelighet markere baksiden av kortene i kortstokken.

Kortene er alle kvadratformet med størrelse 2×2, hvor hjørnet nede til venstre har koordinater $(0, 0)$ og hjørnet øverst til høyre har koordinater $(2, 2)$. Chika tegner bestemte mønste på baksiden av hvert kort, slik at hun ved å inspisere mønsteret, kan vite hvilket tall som er skrevet på fremsiden av kortet. Hun tegner slike mønste ved bruk av følgende prosedyre: Hun plukker så mange ganger hun vil (mulig 0 ganger) to distinkte punkter A og B med heltallskoordinater relativt til nedre venstre hjørnet på kortet, og tegner **rette linjesegmenter** mellom dem.

Chika tegner utelukkende **gyldige** segmenter, altså segmenter mellom to punkter A og B slik at det ikke er noe annet punkt C (ulikt A og B) med heltallskoordinater slik at C ligger på segmentet. For eksempel er segmentet mellom $(0, 0)$ og $(2, 2)$ **ikke gyldig**, da punktet $(1, 1)$ ligger på segmentet. Segmentene mellom $(0, 0)$ og $(1, 1)$ og $(1, 1)$ og $(2, 2)$ er derimot gyldige, og Chika kan tegne dem på samme mønster. Merk også at segmenter ikke har retning: Segmentet fra A til B er **identisk** til både seg selv og segmentet tegnet fra B til A.

Det er viktig at Chika kan kjenne igjen kortene uansett hvordan de er orientert.
Et kort kan roteres 0, 90, 180 eller 270 grader mot klokken med hensyn på den originale orienteringen.

Din oppgave er å hjelpe Chika tegne mønsteret for de q kortene i kortstokken og så gjenkjenne disse kortene senere.

Implementasjon

Dette er en interaktiv oppgave med to faser ***hver fase involverer en seperat eksekvering av programmet ditt***. Du må implementere to funksjoner:

- En `BuildPattern` funksjon som returnerer mønsteret som skal tegnes på baksiden av et gitt kort. Denne funksjonen vil bli kalt q ganger i den første fasen.
- En `GetCardNumber` funksjon som returnerer nummeret til et (mulig rotert) kort med et påtegnet mønster fra første fase av programmet. Denne funksjonen vil bli kalt q ganger i den andre fasen.

Den første funksjonen

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

tar en enkelt parameter n, nummeret som er skrevet på fremsiden av kortet. Du må returnere en `std::vector` som inneholder segmentene Chika tegner som mønster på baksiden av kortet for å kjenne det igjen senere. Et mønster representeres som et `std::pair` av punktet, og et punkt er representert som et `std::pard` (x, y) av heltallskoordinater relativt til nedre venstre hjørnet på kortet, hvor $0 \leq x, y \leq 2$. Alle segmentene Chika tegner må være gyldige og parvis ikke-identiske. Det er garantert at alle q kall til BuildPattern mottar forskjellige verdier for parameteret n.

Etter å ha mottatt alle mønstrene for de q kortene kan *graderen* gjøre gjøre hvilke som helst av følgende operasjoner, vilkårlig antall ganger, på hvert av kortene.

- Rotere hele mønsteret 0, 90, 180 eller 270 grader mot klokken.
- Endre rekkefølgen på segmentene i `std::vector`-representasjonen av mønsteret.
- Endre rekkefølgen på endepunktene av segmentene i mønsteret. (Et segment tegnet fra A til B kan bli gjort om til det ekvivalente segmentet tegnet fra B til A.)

Den andre funksjonen,

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

tar en enkelt parameter p, en `std::vector` av segmenter som beskriver mønsteret tegnet av Chika på baksiden av et kort, basert på returverdien av et tidligere kall til din `BuildPattern`-funksjon. Funksjonen må returnere nummeret n skrevet på forsiden av kortet. Husk at mønsteret p ikke nødvendigvis er i sin originale form som det ble returnert av `BuildPattern`; det kan ha blitt utsatt for de tre operasjonen nevnt over. Det er også mulig at rekkefølgen på kortene er annerledes enn rekkefølgen de ble gitt i første fase, men det er garantert av hvert kort vil bli brukt nøyaktig en gang.

Begrensninger

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ for alle kall til funksjonen `BuildPattern`.
- Merk at det eksisterer algoritmer for å konsturere mønstre slik at en kan kjenne igjen $67\,000\,000$ different cards.

Uttelling

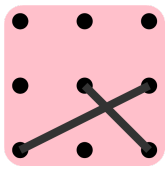
- Deloppgave 1 (2 poeng): $n \leq 2$.
- Deloppgave 2 (9 poeng): $n \leq 25$.
- Deloppgave 3 (15 poeng): $n \leq 1\,000$ og graderen vil **ikke rotere** mønstrene mellom fase 1 og 2. (Graderen **kan** utføre de to andre operasjonene.)
- Deloppgave 4 (3 poeng): $n \leq 16\,000\,000$ og graderen vil **ikke rotere** mønstrene mellom fase 1 og 2. (Graderen **kan** utføre de to andre operasjonene.)
- Deloppgave 5 (24 poeng): $n \leq 16\,000\,000$.
- Deloppgave 6 (18 poeng): $n \leq 40\,000\,000$.
- Deloppgave 7 (29 poeng): Ingen ekstra begrensninger.

Eksempelinteraksjon

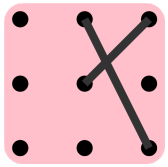
Funksjonskall	Returverdi	Forklaring
Første fase starter.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Vi må lage et mønster for nummeret 3 på kortet med størrelse 2×2. Vi bestemmer oss for å tegne 2 segmenter: - mellom $(0, 0)$ og $(2, 1)$, - mellom $(1, 1)$ og $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Vi må lage et mønster for nummeret 1 på kortet med størrelse 2×2. Vi bestemmer oss for å tegne 1 segment: - mellom $(0, 1)$ og $(0, 0)$.
Første fase slutter.	-	-
Andre fase starter.	-	-
`GetCardNumber( {{{0, 0}, {0, 1}}})`	`1`	Vi får et mønster laget av 1 segment: - mellom $(0, 0)$ og $(0, 1)$. Dette er samme mønster som om vi tegnet segmentet: - mellom $(0, 1)$ og $(0, 0)$. som er nøyaktig det samme segmentet med samme orientering (rotert 0 grader) som vi returnerte på det andre kallet til funksjonen `BuildPattern`. Derfor returnerer vi 1.
`GetCardNumber( {{{1, 1}, {2, 2}}, {{1, 2}, {2, 0}}})`	`3`	Vi får et mønster laget av 2 segmenter: - mellom $(1, 1)$ og $(2, 2)$, - mellom $(1, 2)$ og $(2, 0)$. Dette er mønsteret vi returnerte på det første kallet til `BuildPattern`-funksjonen, rotert med 90-grader mot klokken. Derfor returnerer vi 3.
Andre fase slutter.	-	-

De tre neste bildene representerer, i rekkefølge:

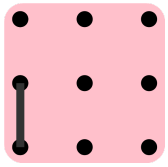
- Mønsteret returnert som output fra det første kallet til `BuildPattern`:



- Mønsteret mottat som parameter av det andre kallet til `GetCardNumber`, som er det første mønsteret rotert 90 grader mot klokken.



- Mønsteret returnert som output av det andre kallet til `BuildPattern`, som også er det samme mønsteret mottatt som paramter av det første kallet til `GetCardNumber` .



Test-grader

Det er gitt en *grader*, `grader.cpp`, i oppgavevedlegget `Cheat.zip`.

Graderen leser et heltall q fra standard input og gjør så følgende steg q ganger:

- Leser et heltall n fra standard input.
- Kaller `BuildPattern(n)` og lagrer returverdien i en variabel p.
- Kaller `GetCardNumber(p)` og skriver ut returverdien til standard input.

Du kan endre test-graderen lokalt om du ønsker.

For å kompilere test-graderen sammen med din løsning, kan du bruke følgende kommando i terminalen:

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

hvor `solution.cpp` er din løsningsfil som skal lastes opp til CMS. For å kjøre programmet med testinput gitt i vedlegget, skriv følgende kommando i terminalen:

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

Merk at, i motsetning til test-graderen, vil den virkelige graderen på CMS utføre den første og andre fasen i seperate eksekusjoner av programmet ditt