# Chika Wants to Cheat

| Nome | Cheat |
|---|---|
| File di input | Task interattivo |
| Output File | Task interattivo |
| Limite di tempo | 2 secondi |
| Limite di memoria | 512 megaottetti |

Chika ha un mazzo di $q$ carte da gioco numerate con vari interi positivi. Vuole giocare ad alcuni giochi con i suoi amici del consiglio studentesco dell'Accademia di Shuchi'in usando queste carte, ma vuole anche vincere, quindi decide di disegnare segretamente sul retro delle carte.

Le carte sono tutte di forma quadrata e di dimensione $2 \times 2$. Chika disegna un certo pattern sul retro di ogni carta, in modo che in seguito possa sapere, guardando il pattern, quale numero è scritto sul davanti della carta. Disegna un tale schema usando la seguente procedura: tante volte quante vuole (possibilmente $0$ volte) sceglie due punti distinti $A$ e $B$ che hanno coordinate intere relative all'angolo in basso a sinistra della carta e disegna un **segmento di linea retta** tra di loro.

Chika disegnerà solo segmenti **validi**, cioè segmenti tra due punti $A$ e $B$ per i quali non esiste un altro punto $C$ (distinti da $A$ e $B$) con coordinate intere che si trova anche esso sul segmento. Ad esempio, il segmento compreso tra $(0, 0)$ e $(2, 2)$ **non è valido** poiché contiene il punto $(1, 1)$, ma segmenti compresi tra $(0, 0)$ e $(1, 1)$ e tra $(1, 1)$ e $(2, 2)$ sono entrambi **validi** e Chika può anche disegnarli entrambi sullo stesso schema. Inoltre, nota che i segmenti non hanno direzione: un segmento disegnato da $A$ a $B$ è **identico** al segmento disegnato da $B$ ad $A$.

È importante sottolineare che Chika vuole assicurarsi di poter riconoscere le sue carte indipendentemente da come vengono ruotate. Una scheda può essere ruotata di $0$, $90$, $180$ o $270$ gradi in senso antiorario rispetto all'orientamento originale.

Il tuo compito è aiutare Chika a disegnare gli schemi per le $q$ carte nel suo mazzo e poi a riconoscere quelle carte in seguito.

# Implementazione

*Questa è un task interattivo con due fasi e **ogni fase coinvolge un'esecuzione separata del tuo programma***. Devi implementare due funzioni:

- Una funzione `BuildPattern` che restituisce il pattern da disegnare sul retro di una determinata carta. Questa funzione sarà chiamata $q$ volte nella prima fase.
- Una funzione `GetCardNumber` che restituisce il numero di una carta (possibilmente ruotata) su cui nella prima fase è stato disegnato un certo pattern. Questa funzione verrà chiamata $q$ volte nella seconda fase.

La prima funzione

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

prende un singolo parametro $n$, il numero che è scritto sulla carta. Devi restituire un `std::vector` contenente i segmenti che descrivono il pattern che Chika disegna sul retro della carta per riconoscerla in seguito. Ogni segmento è rappresentato come una `std::pair` di punti e ogni punto è rappresentato come una `std::pair` $(x, y)$ di coordinate intere rispetto all'angolo inferiore sinistro della scheda, dove $0 \leq x, y \leq 2$. Tutti i segmenti disegnati da Chika devono essere validi e non identici a due a due. È garantito che tutte le chiamate a `BuildPattern` riceveranno valori diversi per il parametro $n$.

Dopo aver ricevuto tutti i pattern per le $q$ carte, il grader può eseguire una qualsiasi delle seguenti operazioni, un numero qualsiasi di volte, su ciascuno dei pattern:

- Ruotare l'intero pattern di $0$, $90$, $180$ o $270$ gradi in senso antiorario.
- Modificare l'ordine dei segmenti nel `std::vector` che rappresenta il pattern.
- Modificare l'ordine degli estremi di un segmento nel pattern. (Un segmento disegnato da $A$ a $B$ può diventare il segmento identico disegnato da $B$ a $A$.)

La seconda funzione,

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

prende un singolo parametro $p$, un `std::vector` di segmenti che descrivono il pattern che viene disegnato da Chika sul retro della carta, in base al valore di ritorno di una precedente chiamata alla tua funzione `BuildPattern`. La funzione deve restituire il numero $n$ scritto sulla parte anteriore della carta. Nota che il pattern $p$ non è necessariamente nella forma originale restituita da `BuildPattern`; potrebbe aver subito una o più delle tre operazioni menzionate sopra. È anche possibile che l'ordine delle carte sia diverso dall'ordine in cui sono state date nella prima fase, ma è garantito che ogni carta è presente esattamente una volta.

## Assunzioni

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ per tutte le chiamate alla funzione `BuildPattern`.
- Nota che esistono algoritmi per costruire pattern tali che sia possibile riconoscere $67\,000\,000$ carte differenti.

## Sottoproblemi

- Subtask 1 (2 punti): $n \leq 2$.
- Subtask 2 (9 punti): $n \leq 25$.
- Subtask 3 (15 punti): $n \leq 1\,000$ e il grader **non ruoterà** i pattern tra la fase 1 e la fase 2. (Il grader **potrebbe** effettuare le altre due operazioni.)
- Subtask 4 (3 punti): $n \leq 16\,000\,000$ e il grader **non ruoterà** i pattern tra la fase 1 e la fase 2. (Il grader **potrebbe** effettuare le altre due operazioni.)
- Subtask 5 (24 punti): $n \leq 16\,000\,000$.
- Subtask 6 (18 punti): $n \leq 40\,000\,000$.
- Subtask 7 (29 punti): Nessuna limitazione aggiuntiva.

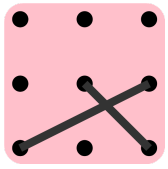
## Esempio di interazione

| Chiamata a funzione | Valore di ritorno | Spiegazione |
|---|---|---|
| Inizia la prima fase. | - | - |
| `BuildPattern(3)` | `{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}` | Dobbiamo creare un pattern per il numero $3$ su una carta di dimensioni $2 \times 2$. Decidiamo di disegnare $2$ segmenti:<br>- tra $(0, 0)$ e $(2, 1)$,<br>- tra $(1, 1)$ e $(2, 0)$. |
| `BuildPattern(1)` | `{{{0, 1}, {0, 0}}}` | Dobbiamo creare un pattern per il numero $1$ sulla carta di dimensioni $2 \times 2$. Decidiamo di disegnare un segmento $1$:<br>- tra $(0, 1)$ e $(0, 0)$. |
| Finisce la prima fase. | - | - |
| Inizia la seconda fase. | - | - |
| `GetCardNumber( {{{0, 0}, {0, 1}}})` | `1` | Otteniamo un pattern composto da 1 segmento:<br>- tra $(0, 0)$ e $(0, 1)$.<br>Questo è lo stesso pattern che otterremmo disegnando il segmento:<br>- tra $(0, 1)$ e $(0, 0)$. |

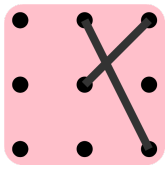
| | | |
|---|---|---|
| | | che è esattamente lo stesso pattern con lo stesso orientamento (ruotato di 0 gradi) che abbiamo restituito alla seconda chiamata alla funzione `BuildPattern`. Pertanto, restituiamo $1$. |
| `GetCardNumber({{{1, 1}, {2, 2}}, {{1, 2}, {2, 0}}})` | `3` | Otteniamo un pattern composto da 2 segmenti:<br>- tra $(1, 1)$ e $(2, 2)$,<br>- tra $(1, 2)$ e $(2, 0)$.<br>Questo è il pattern che abbiamo restituito dalla prima chiamata alla funzeion `BuildPattern`, ruotato di $90$ gradi in senso antiorario. Pertanto, restituiamo $3$. |
| Finisce la seconda fase. | - | - |

Le tre immagini successive rappresentano, nell'ordine:

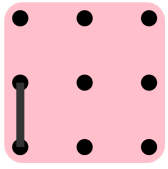
- Il pattern restituito come output dalla prima chiamata a `BuildPattern`:



- Il pattern ricevuto come parametro dalla seconda chiamata a `GetCardNumber`, che è il primo pattern dopo che è stato ruotato di 90 gradi in senso antiorario.



- Il pattern restituito come output dalla seconda chiamata a `BuildPattern`, che è anche lo stesso pattern ricevuto come argomento dalla prima chiamata a `GetCardNumber`.



## Grader di esempio

Il grader di esempio fornito, `grader.cpp`, nell'allegato del task `Cheat.zip`, legge un intero $q$ dallo standard input e esegue i seguenti passaggi $q$ volte:

- Legge un intero $n$ dallo standard input.
- Chiama `BuildPattern(n)` e memorizza il valore restituito in una variabile $p$.
- Chiama `GetCardNumber(p)` e stampa il valore di ritorno sullo standard output.

Se lo desideri, puoi modificare il tuo grader localmente.

Per compilare il grader di esempio con la tua soluzione, puoi utilizzare il seguente comando sul terminale:

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

dove `solution.cpp` è il file della tua soluzione da inviare a CMS. Per eseguire il programma con l'input di esempio fornito nell'allegato, esegui il seguente comando sul terminale:

```
./soluzione < input.txt
```

**Tieni presente che, a differenza del grader di esempio, il vero grader su CMS eseguirà la prima e la seconda fase in esecuzioni separate del tuo programma.**