Chika Wants to Cheat

Nom du problème	Cheat
Fichier d'entrée	Tâche interactive
Fichier de sortie	Tâche interactive
Limite de temps	2 secondes
Limite de mémoire	512 megaoctets

Chika a un jeu de q cartes à jouer sur lesquelles sont inscrits divers entiers positifs. Elle veut utiliser ces cartes pour jouer à certains jeux avec ses amies du conseil des étudiantes de Shuchi'in Academy, mais elle veut aussi gagner, donc elle décide de marquer secrètement le dos des cartes de son paquet.

Les cartes ont toutes une forme carrée de dimensions 2×2, où le coin en bas à gauche a pour coordonnées $(0, 0)$ et le coin en haut à droite a pour coordonnées $(2, 2)$. Chika dessine un certain motif sur le dos de chaque carte, pour qu'elle puissse savoir plus tard, en regardant le motif, quel nombre est écrit sur la face avant de la carte. Elle dessine un tel motif en utilisant la procédure suivante : autant de fois qu'elle le souhaite (potentiellement 0 fois), elle choisit deux points distincts A et B qui ont des coordonnées entières relativement au coin bas gauche de la carte et dessine un **segment de ligne droite** entre eux.

Chika dessinera uniquement des segments **valides**, où un segment valide est un segment entre deux points A et B pour lequel il n'existe aucun un autre point C (distinct de A et de B) avec des coordonnées entières qui se trouve aussi sur le segment. Par exemple, le segment entre $(0, 0)$ et $(2, 2)$ n'est **pas valide** car il contient le point $(1, 1)$, mais les segments $(0, 0)$ et $(1, 1)$ et entre $(1, 1)$ et $(2, 2)$ sont tous les deux **valides**, et Chika peut même les dessiner tous les deux sur le même motif. Notez également que les segments n'ont pas de direction : un segment dessiné de A à B est **identique** à la fois à lui-même et au segment dessiné dans la direction opposée, de B à A.

Il est important pour Chika de pouvoir s'assurer qu'elle reconnaîtra ses cartes même si elles sont tournées. Une carte peut être tournée de 0, 90, 180 ou 270 degrés dans le sens inverse des aiguilles d'une montre relativement à l'orientation initiale.

Votre objectif est d'aider Chika à concevoir les motifs pour les q cartes de son paquet, puis reconnaître ces cartes plus tard.

Implémentation

Ceci est une tâche interactive avec deux étapes, ***chaque étape impliquant une exécution séparée de votre programme***. Vous devez implémenter deux fonctions :

- Une fonction `BuildPattern` qui retourne le motif devant être dessiné sur le dos d'une carte donnée. Cette fonction va être appelée q fois dans la première étape.
- Une fonction `GetCardNumber` qui renvoie le nombre d'une carte (possiblement retournée) qui a le motif donné dessiné lors de la première étape. Cette fonction va être appelée q fois lors de la deuxième étape.

La première fonction

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

prend un unique paramètre n, le nombre écrit sur le devant de la carte. Vous devez retourner un `std::vector` contenant les segments du motif que Chika dessine sur le dos d'une carte pour la reconnaître plus tard. Un segment est représenté par une `std::pair` de points, et un point est représenté par une `std::pair` (x, y) de coordonnées entières, relatives au coin en bas à gauche de la carte, où $0 \leq x, y \leq 2$.. Tous les segments que Chika dessine doivent être valides et non-identiques (selon la définition plus haut dans le sujet). Il est garanti que tous les q appels à BuildPattern reçoivent différentes valeurs pour le paramètre n.

Après avoir reçu tous les motifs pour les q cartes, l'évaluateur peut faire les opérations suivantes, un nombre quelconque de fois, sur chacun des motifs :

- Tourner le motif entier par 0, 90, 180 ou 270 degrés dans le sens inverse des aiguilles d'une montre.
- Modifier l'ordre des segments dans le `std::vector` qui représente le motif.
- Changer l'ordre des extrémités d'un segment dans un motif. (Un segment dessiné de A à B peut devenir le segment identique qui va de B à A.)

La deuxième fonction,

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

prend un unique paramètre p, un `std::vector` de segments décrivant le motif qui est dessiné par Chika au dos d'une carte, selon la valeur de retour d'un appel précédent à votre fonction `BuildPattern`. La fonction doit retourner le nombre n écrit sur le devant de la carte. Pour rappel, le motif p n'est pas forcément dans la forme d'origine telle que retournée par `BuildPattern`; il a pu être modifié par les trois opérations décrites plus tôt. Il est aussi possible que l'ordre des cartes

soit différent de celui donné lors de la première étape, mais il est garanti que chaque carte sera utilisée exactement une fois.

Contraintes

- $1 \leq q \leq 10\,000$.
- $1 \leq n \leq 67\,000\,000$ pour tous les appels à la fonction `BuildPattern`.
- Notez qu'il existe des algorithmes pour construire des motifs tels qu'il soit possible de reconnaître $67\,000\,000$ cartes différentes.

Score

- Sous-tâche 1 (2 points) : $n \leq 2$.
- Sous-tâche 2 (9 points) : $n \leq 25$.
- Sous-tâche 3 (15 points) : $n \leq 1\,000$ et l'évaluateur **ne va pas tourner** les motifs entre les étapes 1 et 2. (L'évaluateur **peut** effectuer les deux autres opérations.)
- Sous-tâche 4 (3 points) : $n \leq 16\,000\,000$ et l'évaluateur **ne va pas tourner** les motifs entre les étapes 1 et 2. (L'évaluateur **peut** effectuer les deux autres opérations.)
- Sous-tâche 5 (24 points) : $n \leq 16\,000\,000$.
- Sous-tâche 6 (18 points) : $n \leq 40\,000\,000$.
- Sous-tâche 7 (29 points) : Aucune contrainte supplémentaire.

Exemple d'interaction

Appel de fonction	Valeur retournée	Explication
La première étape commence.	-	-
`BuildPattern(3)`	`{{{0, 0}, {2, 1}}, {{1, 1}, {2, 0}}}`	Il faut créer un motif pour le nombre 3 sur la carte de taille 2×2. On décide de dessiner 2 segments : - entre $(0, 0)$ et $(2, 1)$, - entre $(1, 1)$ et $(2, 0)$.
`BuildPattern(1)`	`{{{0, 1}, {0, 0}}}`	Il faut créer un motif pour le nombre 1 sur la carte de taille 2×2. On décide de dessiner 1 segment : - entre $(0, 1)$ et $(0, 0)$.
La première étape se termine.	-	-
La deuxième étape commence.	-	-
`GetCardNumber(`	`1`	On a un motif fait d'1 segment :

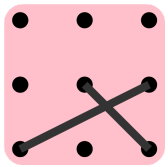
`{{{0, 0}, {0, 1}}})`		- entre $(0,0)$ et $(0,1)$. Cela correspond au même motif que celui obtenu en dessinant le segment : - entre $(0,1)$ et $(0,0)$ Ce qui est exactement le même motif avec la même orientation (tourné de 0 degrés) que celui qui a été retourné par le second appel à la fonction `BuildPattern`. On retourne donc 1.

`GetCardNumber(`
`{{{1, 1}, {2, 2}},`
`{{1, 2}, {2, 0}}})`
	`3`	On a un motif fait de 2 segments :
		- entre $(1,1)$ et $(2,2)$,
		- entre $(1,2)$ et $(2,0)$.
		Ce qui est le motif retourné par le premier appel à la fonction `BuildPattern`, tourné de 90 degrés dans le sens inverse des aiguilles d'une montre. On retourne donc 3.

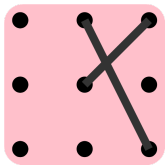
La deuxième étape se termine.	-	-

Les trois images suivantes représentent, dans l'ordre :

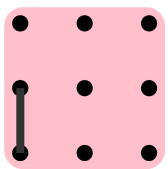
- Le motif retourné par le premier appel à `BuildPattern` :



- Le motif reçu comme paramètre par le second appel à `GetCardNumber`, ce qui est le premier motif après rotation de 90 degrés dans le sens inverse des aiguilles d'une montre.



- Le motif retourné par le deuxième appel à `BuildPattern`, ce qui est aussi le même motif reçu comme argument par le premier appel à `GetCardNumber`.



Exemple d'évaluateur

L'exemple d'évaluateur fourni, `grader.cpp`, dans le dossier joint `Cheat.zip`, lit un entier q depuis l'entrée standard et ensuite va effectuer q fois les étapes suivantes :

- Lire un entier n depuis l'entrée standard.
- Appeler `BuildPattern(n)` et stocker la valeur retournée dans une variable p.
- Appeler `GetCardNumber(p)` et afficher la valeur retournée sur la sortie standard.

Vous pouvez modifier votre évaluateur localement si vous le souhaitez.

Pour compiler l'exemple d'évaluateur avec votre solution, vous pouvez utiliser la commande suivante dans votre terminal :

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

où `solution.cpp` est votre solution à soumettre à CMS. Pour lancer le programme avec l'exemple d'entrée fourni dans le dossier joint, tapez la commande suivante dans votre terminal :

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

Attention : contrairement à l'exemple d'évaluateur, le véritable évaluateur sur CMS va effectuer la première et la deuxième étape sur deux exécutions séparées de votre programme.