Cité idéale (Ideal city)

Comme de nombreux scientifiques et artistes de son temps, Léonard s'est beaucoup intéressé à la planification et conception urbaine. Il avait en tête un modèle de cité idéale : confortable, spacieuse et faisant un usage raisonné des ressources, à l'opposé des villes encombrées et étouffantes du Moyen Âge.

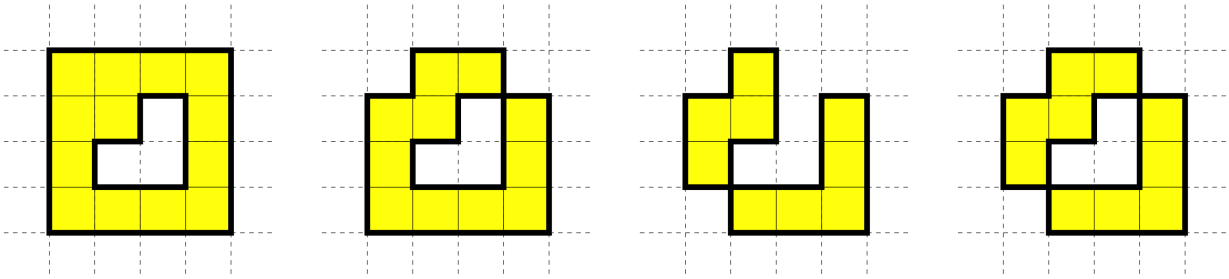
La cité idéale

La cité est constituée de N blocs positionnés sur les cases d'une grille infinie. Chaque case est identifiée par une paire de coordonnées (ligne, colonne). Étant donné une case (i, j), les cases adjacentes sont : (i - 1, j), (i + 1, j), (i, j - 1) et (i, j + 1). Chaque bloc, lorsqu'il est placé sur la grille, couvre exactement une case. Un bloc peut être placé sur la case (i, j) si et seulement si $1 \leq i, j \leq 2^{31} - 2$. Nous utiliserons les coordonnées des cases pour identifier les blocs. Deux blocs sont donc adjacents s'ils sont situés sur des cases adjacentes. Dans une cité idéale, les blocs sont connectés de façon à ce qu'il n'y ait pas de "trous", c'est à dire que les cases doivent satisfaire les deux conditions suivantes.

- Pour tout couple de cases *vides*, il existe toujours au moins un chemin constitué de cases *vides* qui les relie.
- Pour tout couple de cases *non-vides*, il existe toujours au moins un chemin constitué de cases *non-vides* qui les relie.

Exemple 1

Aucune des configurations de blocs ci-dessous ne représente une cité idéale : les deux premières violent la première règle, la troisième viole la seconde règle et la troisième ne respecte aucune des deux règles.



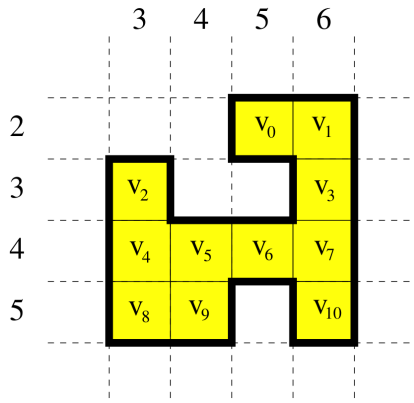
Distance

Lorsque l'on traverse la cité, un *saut* indique que l'on passe d'un bloc à un autre bloc adjacent. Les

cases vides ne peuvent pas être traversées. Soient v_0, v_1, ..., v_{N-1} les coordonnées des N blocs placés sur la grille. Étant donnés deux blocs distincts aux coordonnées v_i et v_j, la distance $d(v_i, v_j)$ est le plus petit nombre de sauts requis pour relier ces deux blocs.

Exemple 2

La configuration ci-dessous représente une cité idéale constituée de N = 11 blocs aux coordonnées $v_0 = (2, 5)$, $v_1 = (2, 6)$, $v_2 = (3, 3)$, $v_3 = (3, 6)$, $v_4 = (4, 3)$, $v_5 = (4, 4)$, $v_6 = (4, 5)$, $v_7 = (4, 6)$, $v_8 = (5, 3)$, $v_9 = (5, 4)$ et $v_{10} = (5, 6)$. Par exemple, $d(v_1, v_3) = 1$, $d(v_1, v_8) = 6$, $d(v_6, v_{10}) = 2$ et $d(v_9, v_{10}) = 4$.



Problème

Votre tâche est d'écrire un programme qui, étant donné une cité idéale, calcule la somme des distances entre toutes les paires de blocs v_i et v_j, pour tout $i < j$. Plus formellement, votre programme doit calculer la somme suivante :

$$\sum d(v_i, v_j), \text{ où } 0 \leq i < j \leq N - 1$$

Vous devez donc implémenter une fonction `DistanceSum`(N, X, Y) qui, étant donnés N et deux tableaux X et Y qui décrivent la cité, calcule la valeur de la somme ci-dessus. Les deux tableaux X et Y sont de taille N. Le bloc i a pour coordonnées (X[i], Y[i]) avec $0 \leq i \leq N - 1$ et $1 \leq X[i], Y[i] \leq 2^{31} - 2$. Étant donné que le résultat peut ne pas tenir sur un entier 32 bits, vous devez calculer ce résultat modulo 1 000 000 000 (un milliard).

Dans l'exemple 2, il y a 11 x 10 / 2 = 55 paires de blocs. La somme des distances entre toutes les paires est de 174.

Sous-tâche 1 [11 points]

Vous pouvez considérer que $N \leq 200$.

Sous-tâche 2 [21 points]

Vous pouvez considérer que $N \leq 2\,000$.

Sous-tâche 3 [23 points]

Vous pouvez considérer que $N \leq 100\,000$.

De plus, les deux conditions suivantes seront satisfaites : étant donné deux cases non-vides i et j telles que X[i] = X[j], toutes les cases entre elles seront aussi non-vides et étant donné deux cases non-vides i et j telles que Y[i] = Y[j], toutes les cases entre elles seront aussi non-vides.

Sous-tâche 4 [45 points]

Vous pouvez considérer que $N \leq 100\,000$.

Détails d'implémentation

Vous devez soumettre exactement un fichier nommé `city.c`, `city.cpp` ou `city.pas`. Ce fichier doit implémenter le sous-programme décrit ci-dessus en utilisant l'une des signatures suivantes.

Programmes C/C++

```
int DistanceSum(int N, int *X, int *Y);
```

Programmes Pascal

```
function DistanceSum(N : LongInt; var X, Y : array of LongInt) : LongInt;
```

Ce sous-programme doit se comporter comme décrit ci-dessous. Vous pouvez bien sûr implémenter d'autres sous-programmes pour votre propre usage. Vos soumissions ne doivent en aucun cas interagir avec les entrées/sorties standards ni aucun autre fichier.

Évaluateur d'exemple

L'évaluateur d'exemple fourni avec l'environnement de la tâche attend en entrée le format suivant :

- ligne 1 : N ;
- ligne 2, ..., N + 1 : X[i], Y[i]

Limite en temps et en mémoire

- Limite en temps : 1 seconde.
- Limite en mémoire : 256 Mio.