Olimpiada Chilena de Informática 2016

Clasificatoria IOI

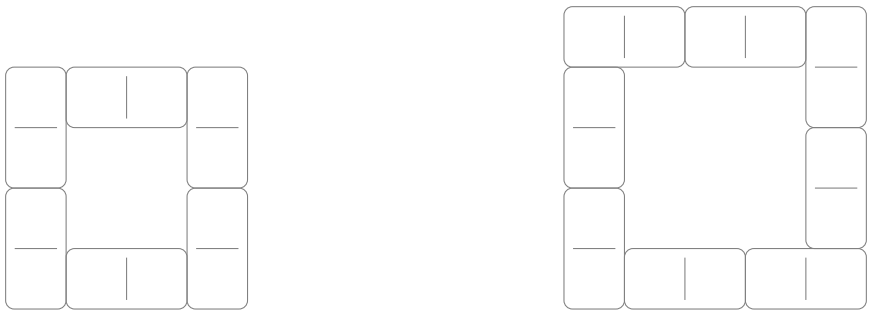
16 de Abril, 2016

Problema A
Dominó

Un juego de fichas de dominó consiste en un conjunto de piezas rectangulares de 2×1, donde cada mitad contiene un número entre 0 y $n-1$. Llamaremos un n-dominó al juego de fichas donde el valor correspondiente es n. Por ejemplo, el juego de dominó normal es un 7-dominó y sus piezas tienen valores entre 0 y 6. Además, en un n-dominó cada combinación de pares de números está presente exactamente una vez (1–3 y 3–1 son la misma pieza). Por ejemplo, un **7-dominó** consiste en **28 piezas**.

Un n-dominó es tan versátil como una baraja de cartas, en cuanto a que ambos permiten jugar una gran variedad de juegos. Uno de estos es el del cuadrado[1]. En el juego del cuadrado se te entrega un n-dominó y un valor k. El objetivo del juego es formar un cuadrado usando todas las piezas del n-dominó, de forma que para cada lado la suma de todos los valores en él sea k. A un cuadrado formado con un n-dominó de forma que los lados sumen k lo llamaremos un cuadrado (n, k).

Para formar un cuadrado válido la cantidad de pieza verticales y horizontales debe ser la misma. A continuación se muestra una imagen de un cuadrado inválido y uno válido. Notar que no para cualquier



(a) Cuadrado inválido (b) Cuadrado de lado 5

Figura 1: Un cuadrado válido y uno inválido.

valor de n es posible formar un cuadrado con un n-dominó. Por ejemplo, el 2-dominó está formado por 3 piezas (0–0, 0–1 y 1–1) y no es posible armar un cuadrado con ellas. Por otro lado, para un n-dominó donde sí es posible formar un cuadrado no necesariamente es posible armar un cuadrado (n, k) para cualquier k. Por ejemplo, para el 7-dominó la suma de los lados siempre será menor que 90, y por lo tanto no es posible formar un cuadrado $(7, 90)$.

Tu objetivo es, entonces, determinar una configuración de piezas para completar el cuadrado (n, k). Para esto, tendrás que implementar algunas funciones que simplificarán enormemente la resolución del problema.

[1] Inspirado en *Matemáticas Recreativas* de Y. I. Perelman

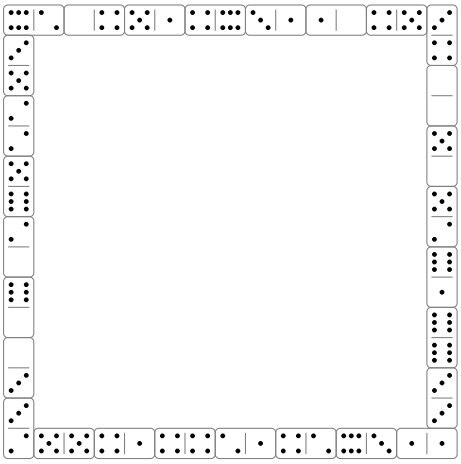


Figura 2: Un cuadrado $(7, 45)$. Notar que cada lado suma 45.

Subtarea 1 (25 pts)

La primera subtarea consiste en implementar la función `cuadrado` que determina para un n si es posible formar un cuadrado con un n-dominó. En esta subtarea no importa la suma de los valores de cada lado, sólo se pide determinar si es posible formar un cuadrado válido usando todas las piezas del n-dominó.

- `bool cuadrado(int n)`
  - `n`: indica el tamaño del n-dominó.
  - *return*: un booleano con valor `true` si se puede formar un cuadrado, y `false` si no.

Para esta subtarea se probará la función `duadrado` con varios casos donde $N \leq 1000$ (25pts).

Subtarea 2 (20 pts)

La subtarea 2 consiste en implementar la función `validar` que dada la especificación de un cuadrado formado con un n-dominó debe verificar que este cumple con la restricción de que todos los lados sumen lo mismo. Esto también implica verificar que las fichas usadas correspondan a las del n-dominó, es decir, cada ficha es usada exactamente una vez. Debes asumir que para el n-dominó entregado es posible formar un cuadrado válido, es decir, `cuadrado(n)` retorna `true`.

- `bool validar(int n, int f, int fichas[])`
  - `n`: indica el tamaño del n-dominó.
  - `f`: indica la cantidad de fichas que tiene el n-dominó.
  - `fichas`: arreglo de enteros de tamaño $2 \times$ `f` donde se especifica el cuadrado. Más adelante se detalla el formato en que el cuadrado debe ser especificado.

- *return*: un booleano con valor `true` si los cuatro lados suman lo mismo, y `false` si no.

Para esta subtarea se probará la función `validar` con varios casos donde $N \leq 1000$ (20pts).

Subtarea 3 y 4

Para las subtareas 3, 4 y 5, debes implementar la función `construir` que para un n y un k dados construye un cuadrado (n, k) en caso de ser posible. Debes asumir que para el n-dominó entregado es posible formar un cuadrado válido, es decir, `cuadrado(n)` retorna `true`. No obstante, puede que ser posible que para el valor de k no sea posible formar un cuadrado (n, k). Si es posible formar un cuadrado (n, k) la función debe retornar `true`, en caso contrario debe retornar `false`.

- `bool construir(int n, int k, int f, int fichas[])`
  - `n, k`: valores que indican el cuadrado (n, k) que se quiere construir.
  - `f`: indica la cantidad de fichas que tiene el n-dominó.
  - `d`: arreglo de enteros de tamaño $2 \times$ `f` donde debes guardar la especificación del cuadrado (n, k). Más adelante se detalla la forma de especificar el cuadrado. Tu función debe llenar este arreglo. Su contenido será revisado solo en el caso en que la función retorne `true`.
  - *return*: un booleano con valor `true` si se puede formar un cuadrado (n, k), y `false` si no.

Para la subtarea 3 se probará la función `validar` con varios casos donde $N \leq 17$ (20pts). Para la subtarea 4 se probará la función `validar` con varios casos donde $N \leq 10000$ (25pts).

Convención sobre el formato del arreglo *d* La función `validar` recibe como parámetro un arreglo de enteros `fichas` donde se especifica un cuadrado formado con las fichas de un n-domino. De la misma forma `construir` debe llenar el arreglo `fichas` para especificar el cuadrado que construye. A continuación se detalla la forma en que un cuadrado es especificado en el arreglo `fichas`.

Si f es la cantidad de fichas de un n-dominó el arreglo `fichas` es de tamaño $2 \times f$. Por ejemplo, como un 7-dominó tiene 28 fichas puede ser guardado en un arreglo de tamaño 56. El arreglo es tal que dos posiciones seguidas corresponden a una ficha. La posición 0 y la 1 corresponden a la primera ficha, la 2 y la 3 a la segunda ficha y así hasta la última ficha que corresponde a las posiciones $2 \times f - 2$ y $2 \times f - 1$ en el arreglo. Las fichas en el arreglo se especifican en sentido horario partiendo desde la esquina superior izquierda. Por ejemplo, el cuadrado mostrado en la Figura 2 puede ser guardada un arreglo de la siguiente manera.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
6	2	0	4	5	1	4	6	3	1	1	0	4	5	3	4	0	0	...	1	4	5	5	2	3	3	0	0	6	0	2	6	5	2	2	5	3

Problema B
El camino del dragón

Nuestro héroe Olon-sonkú debe recorrer el camino del dragón para llegar al planeta de Jorgesama y aumentar su poder de pelea. Desgraciadamente, el camino del dragón no es en línea recta, sino que es un camino infinito conocido como la curva del dragón de Highway.

Si bien nuestro amigo Olon-sonkú tiene una gran fuerza física, su inteligencia es bastante escasa. Incluso los cálculos aritméticos más simples son un gran desafío para él. Es por esto que se encuentra en un gran problema pues para lograr su cometido debe calcular las coordenadas del planeta del gran Jorgesama quien se encuentra en algún lugar del camino del dragón. Por fortuna su amiga Bulnelman planea ayudarlo a realizar esta tarea. Bulnelman estudió detenidamente el camino del dragón y descubrió que este puede ser descrito de manera simple.

El verdadero camino del dragón es un camino infinito, pero a continuación hablaremos de varios caminos finitos que llamaremos también caminos del dragón. Para describir un camino del dragón se denotará con una A la acción de avanzar un paso, con una R la acción de rotar 90° grados a la derecha y con una L la acción de rotar 90° a la izquierda. Un camino del dragón corresponde a una secuencia de estos símbolos que representan las acciones que hay que realizar para recorrerlo. No obstante, no cualquier secuencia formada con estas letras corresponde a un camino del dragón válido. Las secuencias válidas son generadas por un sistema de reescritura. Llamaremos secuencias del dragón a las secuencias generadas por este sistema.

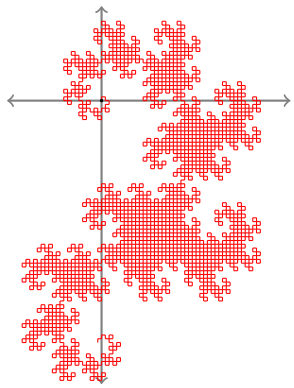


Figura 1: Primeros $2^{12} - 1$ pasos de la curva del dragón

Un sistema de reescritura consiste en una serie de reglas para transformar una secuencia. Cada regla contiene un símbolo a la izquierda y una secuencia a la derecha. Aplicar una regla significa que cada aparición del símbolo de la izquierda debe ser reemplazada por la secuencia de la derecha. Las reglas para generar secuencias del dragón usan dos símbolos auxiliares a y b. Estos símbolos no representan acciones y solo son usados para generar las secuencias. Las reglas correspondientes al camino del dragón son las siguientes.

$$a \to aRbAR \qquad b \to LAaLb$$

Para generar secuencias del dragón hay que partir con la secuencia Aa y aplicar iterativamente estas reglas de reescritura. A continuación se muestra la aplicación de las reglas en tres iteraciones consecutivas. En cada paso ambas reglas son aplicadas simultáneamente.

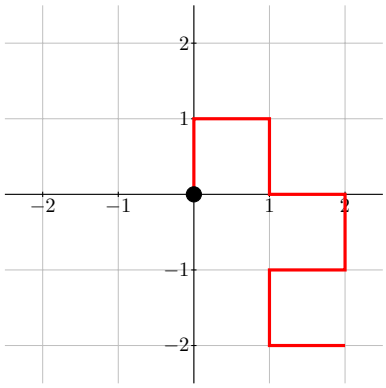
$$Aa \to AaRbAR \to AaRbARRLAaLbAR \to AaRbARRLAaLbARRLAaRbARLLAaLbAR$$

En el primer paso el símbolo a es reemplazado en Aa por la secuencia $aRbAR$ resultando $AaRbAR$. A continuación se reemplaza simultáneamente la a por $aRbAR$ y la b por $LAaLB$ dando como resultado la secuencia $AaRbARRLAaLbAR$. Finalmente después de la tercera iteración se obtiene la secuencia $AaRbARRLAaLbARRLAaRbARLLAaLbAR$. Notar que después de cada iteración la secuencia resultante es una extensión de la anterior. Esto significa que después de cada iteración la secuencia resultante se acerca más al camino del dragón real.

Cada una de las secuencias generadas en el proceso da origen a una secuencia del dragón que corresponde a la secuencia generada eliminando las ocurrencias de a y b. Por ejemplo, las secuencias A, $ARAR$, $ARARRLALAR$ y $ARARRLALARRLARARLLALAR$ son las secuencias del dragón originadas a partir de las primeras tres iteraciones.

Cada una de las secuencias del dragón corresponde a una serie de acciones que hay que hacer para recorrer el camino correspondiente. Para determinar la posición en que queda Olon-sonkú después de realizar las acciones supondremos que siempre comienza en la posición $(0, 0)$ mirando hacia el norte. A continuación se muestra un ejemplo donde se ejecutan las acciones para la secuencia $ARARRLALARRLARARLLALAR$ junto a una figura de ilustración:

- Comenzar en $(0, 0)$ mirando al norte
- A: Avanzar, llega a $(0, 1)$ mirando todavía hacia el norte.
- RA: Rotar a la derecha y avanzar, llega a $(1, 1)$ y mirando hacia el este.
- $RRLA$: Rotar dos veces a la derecha, una vez a la izquierda y luego avanzar, llega a $(1, 0)$ y mirando hacia sur.
- LA: Llega a $(2, 0)$ mirando hacia el este.
- $RRLA$: Llega a $(2, -1)$ mirando hacia el sur.
- RA: Llega a $(1, -1)$ mirando hacia el oeste.
- $RLLA$: Llega a $(1, -2)$ mirando hacia el sur.
- LAR: Termina en $(2, -2)$ mirando hacia el sur.



Diremos que un recorrido del dragón tiene largo N si se ejecuta la acción avanzar N veces. Por ejemplo, el recorrido descrito anteriormente tiene largo $N = 8$. Notar que para hacer un recorrido de largo N hay que considerar una secuencia del dragón que tenga al menos N acciones avanzar. Por ejemplo si se quiere hacer un recorrido de largo $N = 5$ hay que considerar la secuencia $ARARRLALARRLARARLLALAR$ y ejecutar las primeras 5 acciones avanzar. Esto dejaría a Olon-sonkú en la posición $(2, -1)$.

Supón que Bulnelman sabe que el planeta de Jorgesama se encuentra al final del recorrido del dragón de largo N. ¿Puedes ayudarnos a saber cual es la posición exacta del planeta del gran Jorgesama?

Entrada

La entrada consiste en una linea con un único entero positivo N. Tu programa debe calcular las coordenadas del final del recorrido del dragón de largo N.

Salida

Debes imprimir una única línea con dos enteros x e y separados por un espacio. Estos enteros corresponden a las coordenadas del planeta del gran Jorgesama.

Subtareas y Puntaje

10 puntos Se probarán varios casos donde $1 \leq N \leq 8$.

20 puntos Se probarán varios casos donde $1 \leq N \leq 100$.

40 puntos Se probarán varios casos donde $1 \leq N \leq 10^5$.

30 puntos Se probarán varios casos donde $1 \leq N \leq 10^{15}$.

Nota: en la última subtarea el entero N debe ser leído en una variable de tipo long long.

Ejemplos de Entrada y Salida

Entrada de ejemplo	**Salida de ejemplo**
`5`	`2 -1`

Entrada de ejemplo	**Salida de ejemplo**
`8`	`2 -2`

Entrada de ejemplo	**Salida de ejemplo**
`50`	`-5 -1`

Entrada de ejemplo	**Salida de ejemplo**
`500`	`18 16`

Entrada de ejemplo	**Salida de ejemplo**
1234567890	-38671 11005

Entrada de ejemplo	**Salida de ejemplo**
1000000000000000	25747840 -5785984

Problema C
La Gran Fila

Cada cuatro años la Organización de Cine Independiente (OCI) destina un día para que sus integrantes puedan devolver las películas que han arrendado. Como esta instancia se organiza solo cada cuatro años, la fila que se arma cada vez es extremadamente larga. Es por esto que las personas en la fila deben esperar horas hasta poder ser atendidas.

Dependiendo de la cantidad de películas que tiene que devolver, cada persona tendrá un tiempo de atención distinto. Esta información es conocida de antemano. El tiempo que una persona debe esperar en la fila es igual a la suma de los tiempos de atención de las personas delante de ella.

Este año toda la gente ha llegado muy temprano y se encuentra formada en una gran fila fuera de las dependencias de la OCI. Antes de abrir las puertas, un trabajador de la OCI anuncia que este año será posible atender a los asistentes paralelamente en dos filas y por lo tanto es necesario formar una nueva. Para evitar conflictos, el trabajador propone que en el mismo orden en que están formados, cada persona elija si quedarse en la fila actual o cambiarse al final de la nueva fila. Cada persona quiere minimizar el tiempo que estará en la fila y se cambiará de fila solo si eso significa que esperará estrictamente menos tiempo. El tiempo en que las personas se cambian de fila es irrelevante. Tu tarea es encontrar para cada persona cuanto tiempo tendrá que esperar en su fila luego de que las dos filas estén armadas.

Entrada

La entrada consiste en dos líneas. La primera línea contiene un entero N correspondiente al número de personas. La siguiente línea contiene N enteros describiendo la fila inicial. Cada número describe a una persona. Las personas son identificadas con un número de 1 a N en este mismo orden. El número i-ésimo corresponde al tiempo de atención de la persona número i.

Salida

La salida consiste en varias líneas. Cada línea describe a un persona luego de que las dos filas se hayan formado. La línea i-ésima debe contener el tiempo que deberá esperar la persona número i en su fila final, es decir, la suma de todos los tiempos de atención de las personas delante de ella.

Subtareas y Puntaje

40 puntos Se probarán varios casos donde el tiempo de atención de todas las personas es igual a 1 y $N \leq 1000$.

60 puntos Se probarán varios casos donde $N \leq 1000$ y no hay restricciones adicionales.

Ejemplos de Entrada y Salida

Entrada de ejemplo	Salida de ejemplo
5 4 3 2 5 1	0 0 3 4 5

Entrada de ejemplo	Salida de ejemplo
7 4 3 3 2 5 1 3	0 0 3 4 6 6 7

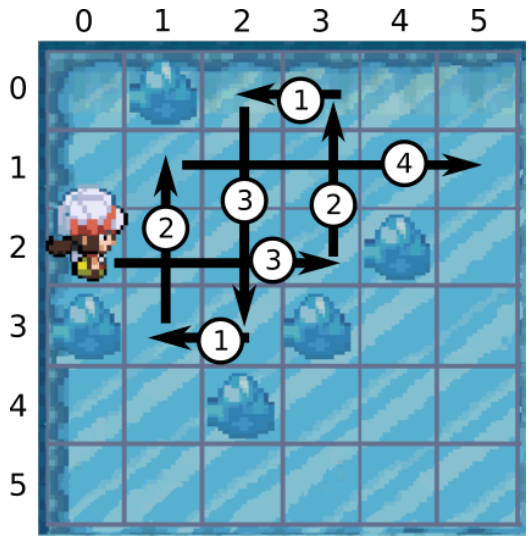
Problema D
La cueva de hielo

Terry es una chica a la que le gustan los desafíos. En este momento ella se encuentra participando en una difícil competencia que consiste en completar complejos laberintos de la cueva de hielo de cierta franquicia de criaturas de bolsillo.

Un laberinto consiste en una grilla de $N \times N$ casillas. Cada casilla es identificada por sus coordenadas x e y ($0 \leq x, y \leq N-1$). Algunas casillas en el laberinto están vacías mientras que otras pueden contener una roca. Además dos casillas especiales son identificadas como la casilla inicial y final. Estas casillas están siempre vacías, es decir, no contienen rocas.

El objetivo es mover a un personaje por el laberinto llevándolo desde la casilla inicial hasta la final. Cada vez que el personaje se mueve a una casilla contigua tarda una unidad de tiempo. Lo complejo de los laberintos es que el piso es de hielo y por lo tanto el personaje no puede moverse libremente. Cada vez que el personaje comienza a moverse en una dirección debe continuar moviéndose en esa dirección hasta chocar con una roca o con el borde del laberinto. Notar que no basta con que el personaje pase por sobre la casilla final, este tiene que quedar detenido en esa posición.

La figura de más abajo muestra un laberinto y un posible camino. El personaje comienza en la casilla $(0, 2)$ y el objetivo es llevarlo a la casilla $(5, 1)$. En primer lugar, el personaje se mueve hacia la derecha hasta chocar con la piedra en la casilla $(3, 2)$ quedando detenido en la casilla $(3, 2)$. Este movimiento tarda 3 unidades de tiempo. A continuación se mueve hacia arriba hasta chocar con el borde superior quedando detenido en la casilla $(3, 0)$ y tardando 2 unidades de tiempo. Los siguientes movimientos llevan al personaje a las casillas $(2, 0), (2, 3), (1, 3), (1, 1)$ y finalmente a la casilla $(5, 1)$. El tiempo total ocupado en estos movimientos es $3 + 2 + 1 + 3 + 1 + 2 + 4 = 16$.



Terry es una chica muy hábil, pero siempre está ocupada organizando eventos y esta vez no ha tenido

tiempo de entrenar para resolver los laberintos. Terry siempre está disponible para ayudar a quién lo necesite. Ahora es momento de que tú la ayudes resolviendo estos laberintos.

Subtarea 1 (50pts)

La primera subtarea consiste en determinar si es posible llevar al personaje desde la casilla inicial a la final. En esta subtarea no es importante el tiempo que toma mover al personaje.

- `bool posible(int N, bool rocas[100][100], int xi, int yi, int xf, int yf)`

  Esta función debe determinar si es posible para el laberinto especificado llevar al personaje desde la posición inicial a la final. Si es posible la función debe retornar `true`, en caso contrario debe retornar `false`.

  - `N`: Tamaño de la grilla.
  - `rocas`: Arreglo de dos dimensiones especificando las casillas del laberinto que contienen una roca. Si `rocas[x][y]` guarda el valor `true` significa que hay una roca en la casilla (x, y). En aso contrario, si `rocas[x][y]` guarda el valor `true` significa que la casilla (x, y) está vacía. Este arreglo es de tamaño 100×100, pero solo las posiciones correspondientes a las primeras $N \times N$ casillas tienen información relevante.
  - `xi`: Coordenada x de la posición inicial.
  - `yi`: Coordenada y de la posición inicial.
  - `xf`: Coordenada x de la posición final.
  - `yf`: Coordenada y de la posición final.

Para esta subtarea se probarán esta función con varios casos donde $10 \leq N \leq 100$ (50pts).

Subtarea 2 (50pts)

En esta subtarea debe encontrarse el mínimo tiempo en que es posible llevar al personaje desde la posición inicial a la final.

- `int minimo(int N, bool rocas[100][100], int xi, int yi, int xf, int yf)`

  Esta función debe retornar un entero correspondiente a la mínima cantidad de tiempo en que es posible llevar al personaje desde la posición inicial a la final. En caso de no ser posible la función debe retornar -1. Los parámetros de la función son los mismos que los de `posible`.

Para esta subtarea se probarán esta función con varios casos donde $10 \leq N \leq 100$ (50pts).

Detalles de implementación

Debes enviar exactamente un archivo, llamado `hielo.cpp`. Este archivo deberá implementar todas las funciones descritas. Si solo haz implementado una de las funciones la otra debe también estar presente y puedes dejarla vacía. El archivo `hielo.cpp` debe también incluir el header `hielo.h`.

Grader de Ejemplo

Se provee un *grader* junto algunos archivos de prueba para que puedas testear tu solución. El grader lee de la entrada estándar en el siguiente formato.

La primera línea contiene dos enteros T y N correspondientes a la subtarea que se quiere resolver y el tamaño de la grilla. El número T puede ser 1 o 2. La siguiente linea contiene cuatro enteros xi, yi, xf y yf. Correspondientes a las coordenadas de la casilla inicial y la final. Las siguientes N líneas contienen la descripción de las posiciones de las rocas en la grilla. Cada línea contiene N enteros separados por espacio. Estos enteros pueden ser 0 o 1. Un valor igual a 0 significa que la casilla no contiene una roca y un valor igual a 1 significa que la casilla si contiene una roca.