Problema
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 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 tiene un tiempo de atención distinto. 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 que se está formando. Cada persona quiere minimizar el tiempo que estará en su fila, y se cambiará a la nueva 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 y cada persona es identificada 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

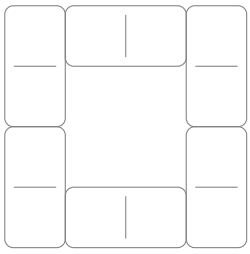
Problema
Dominó

Un juego de dominó tradicional consiste en un conjunto de fichas rectangulares de 2×1, donde cada mitad contiene un número entre 0 y 6. Cada combinación de pares de números aparece exactamente una vez, pero sin considerar el orden. Por ejemplo, la ficha 1–6 es la misma que la 6–1. Esto da un total de 28 fichas para el dominó tradicional.

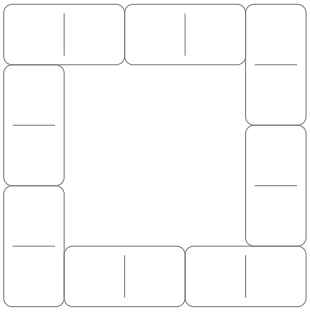
Dado un entero n mayor que cero, el juego de dominó puede ser generalizado para que cada pieza contenga números entre 0 y $n-1$. En este dominó generalizado cada combinación de pares de números también aparece exactamente una vez. Por ejemplo para $n = 2$ las fichas son 0–0, 0–1 y 1–1. Llamaremos un n-dominó al juego donde cada ficha contiene números entre 0 y $n-1$, y cada combinación de pares de números aparece exactamente una vez. Por ejemplo, el dominó tradicional es un 7-dominó. Por otro lado, el conjunto (0–0, 0–2, 2–2) está formado por fichas con valores entre 0 y 2, pero no conforman un 3-dominó, pues hay fichas que faltan.

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. En el juego del cuadrado se te entrega un n-dominó y un valor k. El objetivo del juego es construir un cuadrado usando todas las fichas que conforman el n-dominó, de forma que para cada lado la suma de todos los valores en las fichas sea k. A un cuadrado construido con todas las fichas de un n-dominó en que todos los lados sumen k lo llamaremos un cuadrado (n, k).

Para construir un cuadrado válido la cantidad de fichas verticales y horizontales debe ser la misma. A continuación se muestra una imagen de un cuadrado inválido y uno válido. El cuadrado de la izquierda es inválido pues los lados horizontales están formados por una ficha completa y dos mitades, mientras que los verticales por dos fichas completas. El cuadrado de la derecha es válido y cada lado, horizontal o vertical, está formado por dos fichas completas y una mitad. Notar que todas las fichas están contenidas en un solo lado salvo por las fichas de las esquinas que pertenecen a dos lados simultáneamente.



Cuadrado inválido

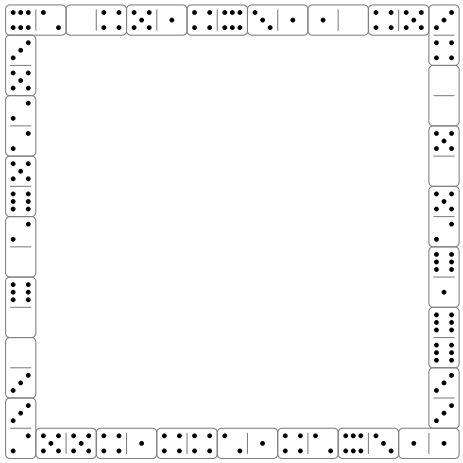


Cuadrado válido

Para un valor n puede ser el caso que dada la cantidad de fichas que conforman un n-dominó sea imposible construir un cuadrado válido. Por ejemplo, el 2-dominó está formado por 3 fichas (0–0, 0–1 y 1–1) y no es posible construir un cuadrado válido con ellas. De hecho el primer n-dominó con el cuál es posible construir un cuadrado válido es el 7-dominó. Por otro lado, para un n-dominó donde sí es

posible construir un cuadrado válido no necesariamente es posible construir 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 $(7, 90)$.

A continuación se muestra la imagen de un cuadrado formado con un 7-dominó donde la suma de cada lado es 45, es decir, es un cuadrado $(7, 45)$. Notar que los números en las esquinas deben ser considerados dos veces. Por ejemplo, el 3 de la ficha 3–4 es contabilizado tanto en la suma del lado superior como en la suma del lado derecho. Este es uno de los cuadrados $(7, 45)$ posibles. En general, para un n y un k existen varias formas de construir un cuadrado (n, k).



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

Dado un valor n y un valor k el objetivo de este problema es construir un cuadrado (n, k). Para esto, tendrás que implementar algunas funciones que simplificarán enormemente la resolución del problema.

Subtarea 1 (25 pts)

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

- `bool cuadrado(int n)`
  - `n`: indica el tamaño del n-dominó.
  - `return`: la función debe retornar `true` si es posible construir un cuadrado válido o `false` en caso contrario.

Restricciones

- Para esta subtarea se probará la función `cuadrado` con varios casos donde $0 < n \leq 1000$ (25pts).

Subtarea 2 (35 pts)

La subtarea 2 consiste en implementar la función `verificar` que dada la especificación de un cuadrado construido con fichas de un n-dominó debe verificar que todos los lados de este sumen lo mismo. La función también debe verificar que todas las fichas que conforman el n-dominó sean usadas, es decir, no hay fichas que falten o que se repitan. En esta subtarea para el n-dominó entregado siempre será posible formar un cuadrado válido, es decir, `cuadrado(n)` retorna `true`.

- `bool verificar(int n, int f, int fichas[])`
  - `n`: indica el tamaño del n-dominó.
  - `f`: indica la cantidad de fichas que conforman el n-dominó.
  - `fichas`: arreglo de enteros de tamaño $2 \times$ `f` donde se especifica el cuadrado que hay que verificar. Más adelante se detalla el formato en que el cuadrado es especificado.
  - `return`: la función debe retornar `true` si todos los lados del cuadrado especificado suman lo mismo y todas las fichas que conforman el n-dominó son utilizadas. En caso contrario debe retornar `false`.

Restricciones

- Para esta subtarea se probará la función `verificar` con varios casos donde $0 < n \leq 1000$ (35pts).

Subtareas 3 y 4 (40 pts)

Para las subtareas 3 y 4, debes implementar la función `construir` que recibe un valor n y un valor k, y construye un cuadrado (n, k) en caso de ser posible. En esta función parar el n-dominó entregado siempre será posible formar un cuadrado válido, es decir, `cuadrado(n)` retorna `true`. No obstante, puede ser el caso 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 conforman un n-dominó.
  - `fichas`: 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 con la descripción de algún cuadrado (n, k). Su contenido será revisado sólo en el caso de que la función retorne `true`.
  - `return`: la función debe retornar `true` si es posible formar un cuadrado (n, k) o `false` en caso contrario.

Restricciones

- Para la subtarea 3 se probará la función `verificar` con varios casos donde $n = 7$ (20pts).
- Para la subtarea 4 se probará la función `verificar` con varios casos donde $7 < n \leq 1000$ (20pts).

Formato del arreglo *fichas*

La función `verificar` recibe como parámetro un arreglo de enteros `fichas` donde se especifica un cuadrado construido con las fichas de un n-dominó. 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 que conforman un n-dominó, el arreglo `fichas` es de tamaño $2 \times f$. Por ejemplo, un cuadrado construido con un 7-dominó debe ser especificado en un arreglo de tamaño 56, pues un 7-dominó está conformado por 28 fichas. En el arreglo una ficha es almacenada en dos posiciones consecutivas. 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 (7,45) mostrado en la figura de más arriba puede ser almacenado en 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

Detalles de implementación

Debes enviar exactamente un archivo, llamado `domino.cpp`. Este archivo debe contener todas las funciones descritas. Si solo has implementado una de las funciones las otras deben también estar presente y puedes dejarlas vacías. El archivo `domino.cpp` debe incluir el header `domino.h`.

Grader de Ejemplo

Se provee un *grader* junto con 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 un entero T correspondiente a la subtarea que se quiere resolver. El número T puede ser 1, 2, 3 o 4.

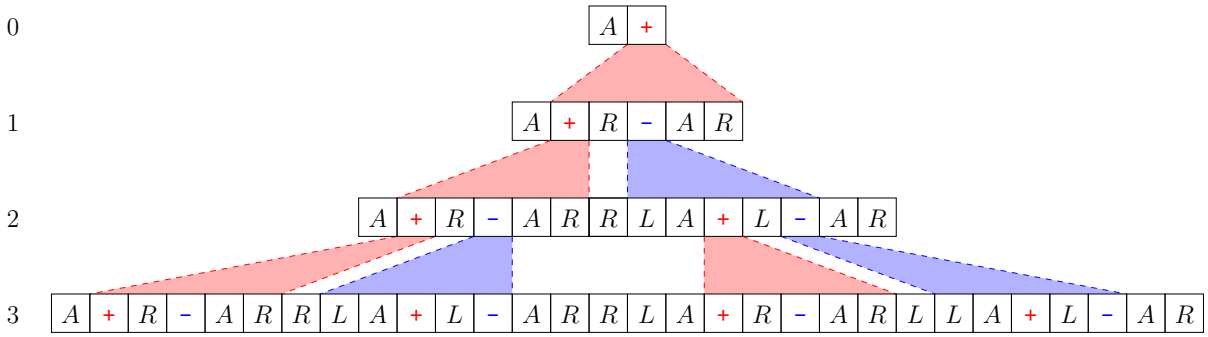
- Si T es 1, solo sigue una línea, que contiene un entero n correspondiente al tamaño del n-dominó.
- Si T es 2, siguen dos líneas. La primera contiene dos enteros n y f, correspondientes al tamaño y el número de fichas del n-dominó respectivamente. La segunda línea contiene $2 \times f$ enteros entre 0 y n, los valores del arreglo `fichas` en el formato descrito anteriormente.
- Si T es 3 o 4, sigue una sola línea. Esta línea contiene tres enteros n, k y f, donde n indica el tamaño del n-dominó, k indica cuanto deben sumar los lados del cuadrado que se pide construir, y f es el número de fichas que conforman el n-dominó.

Problema
El camino del dragón

Nuestro héroe Olon-sonkú debe llegar al planeta del gran maestro Jorgesama para poder aumentar su poder de pelea. Si supiera la posición exacta del planeta podría usar la teletransportación para llegar. Lamentablemente Olon-sonkú no sabe exactamente dónde está el planeta y sólo sabe que este se encuentra en algún punto del *camino del dragón*, un camino infinito de intrincada estructura. Por fortuna, su astuta amiga Bulnelman lleva años estudiando este camino, y ha descubierto que puede ser recorrido de manera simple mediante las *secuencias del dragón* que describen movimientos que se deben seguir para avanzar por el camino infinito.

Para describir una *secuencia del dragón* denotaremos 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. Las secuencias del dragón son descritas usando secuencias de estas letras. No obstante, no cualquier secuencia corresponde a una secuencia del dragón válida. Las secuencias válidas son generadas por un sistema de *reemplazos* que usa dos símbolos especiales, "+" y "−", y que siguen las siguientes reglas. Un "+" se debe reemplazar por la secuencia "+ R − A R" y un "−" se debe reemplazar por la secuencia "L A + L −", lo que se muestra en el siguiente esquema.

Para generar secuencias del dragón hay que comenzar con la secuencia "A +" y aplicar iterativamente las reglas de reemplazo. A continuación se muestra un esquema con la aplicación de las reglas en tres iteraciones consecutivas iniciando en la secuencia "A +".



Comenzando con "A +", en el primer paso el símbolo "+" es reemplazado por "+ R − A R" resultando la secuencia "A + R − A R" que se ve en el nivel 1 en el diagrama. A continuación en esa secuencia se reemplazan "+" por "+ R − A R" y "−" por "L A + L −" dando como resultado la secuencia "A + R − A R R L A + L − A R" en el nivel 2 del diagrama. Después de la tercera iteración se obtiene la secuencia que se ve en el nivel 3 del diagrama y que corresponde a

"A + R − A R R L A + L − A R R L A + R − A R L L A + L − A R".

Finalmente, para generar las secuencias del dragón, debemos eliminar los "+" y "−" de las secuencias generadas en el anterior proceso. El siguiente diagrama muestra las secuencias del dragón generadas a partir de las primeras tres iteraciones.

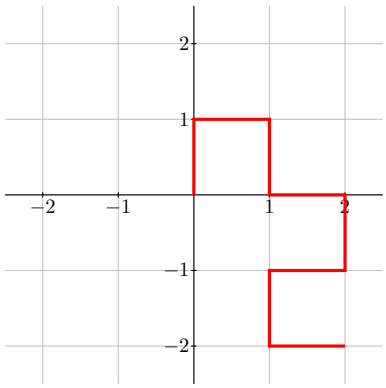
0	A
1	$A\ R\ A\ R$
2	$A\ R\ A\ R\ R\ L\ A\ L\ A\ R$
3	$A\ R\ A\ R\ R\ L\ A\ L\ A\ R\ R\ L\ A\ R\ A\ R\ L\ L\ A\ L\ A\ R$

Notar que después de cada iteración la secuencia resultante es una extensión de la anterior. Esto significa que después de cada iteración nos acercamos más al camino del dragón real. Como lo notó Bulnelman, si siguiéramos con el proceso hasta el infinito obtendríamos una secuencia que define la forma de avanzar por el camino del dragón completo.

Bulnelman tiene el problema de Olon-sonkú casi resuelto, pues sus investigaciones le han permitido determinar que el planeta de Jorgesama se encuentra avanzando exactamente N pasos en el camino del dragón. Lo único que necesitamos ahora es saber dónde quedaría exactamente una persona que avanza N pasos por el camino, y es aquí donde tu ayuda se hace imprescindible.

Supondremos que para recorrer el camino del dragón se comienza siempre en la posición $(0, 0)$ mirando hacia el norte. A continuación se muestra un ejemplo donde se ejecutan las acciones para la secuencia "$A\ R\ A\ R\ R\ L\ A\ L\ A\ R\ R\ L\ A\ R\ A\ R\ L\ L\ A\ L\ A\ R$" junto a una figura de ilustración:

- Comenzamos en $(0, 0)$ mirando al norte
- A: Avanzar, llega a $(0, 1)$ mirando todavía hacia el norte.
- $R\ A$: Rotar a la derecha y avanzar, llega a $(1, 1)$ y mirando hacia el este.
- $R\ R\ L\ A$: Rotar dos veces a la derecha, una vez a la izquierda y luego avanzar, llega a $(1, 0)$ y mirando hacia sur.
- $L\ A$: Llega a $(2, 0)$ mirando hacia el este.
- $R\ R\ L\ A$: Llega a $(2, -1)$ mirando hacia el sur.
- $R\ A$: Llega a $(1, -1)$ mirando hacia el oeste.
- $R\ L\ L\ A$: Llega a $(1, -2)$ mirando hacia el sur.
- $L\ A\ R$: Termina en $(2, -2)$ mirando hacia el sur.



Nota que en el recorrido anteriormente descrito se avanzaron $N = 8$ pasos. Esencialmente para una secuencia del dragón, la cantidad de pasos que se avanza equivale a la cantidad de veces que la letra

"A" aparece en la secuencia. Nota además que para seguir N pasos del camino del dragón, se debe considerar una secuencia con al menos N letras "A". Por ejemplo, para avanzar 5 pasos, hay que considerar la secuencia "$A\ R\ A\ R\ R\ L\ A\ L\ A\ R\ R\ L\ A\ R\ A\ R\ L\ L\ A\ L\ A\ R$" y ejecutar las primeras 5 acciones avanzar, lo que nos dejaría en la posición $(2, -1)$. Si tienes curiosidad acerca de la forma del mítico camino del dragón, en la figura de la siguiente página se muestra una porción más grande del camino, una en donde se han avanzado $2^{12} - 1 = 4.095$ pasos.

Olon-sonkú ya no puede esperar para teletrasportarse done Jorgesama, sólo necesita saber las coordenadas. Por su parte Bulnelman ya sabe la cantidad de pasos que habría que avanzar en el camino del dragón para llegar al planeta. Ahora sólo falta tu parte.

Entrada

La entrada consiste en una línea con un único entero positivo N, que corresponde a la cantidad de pasos que hay que avanzar en el camino del dragón para llegar al planeta de Jorgesama.

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 $1 \leq N \leq 100$.

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

35 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

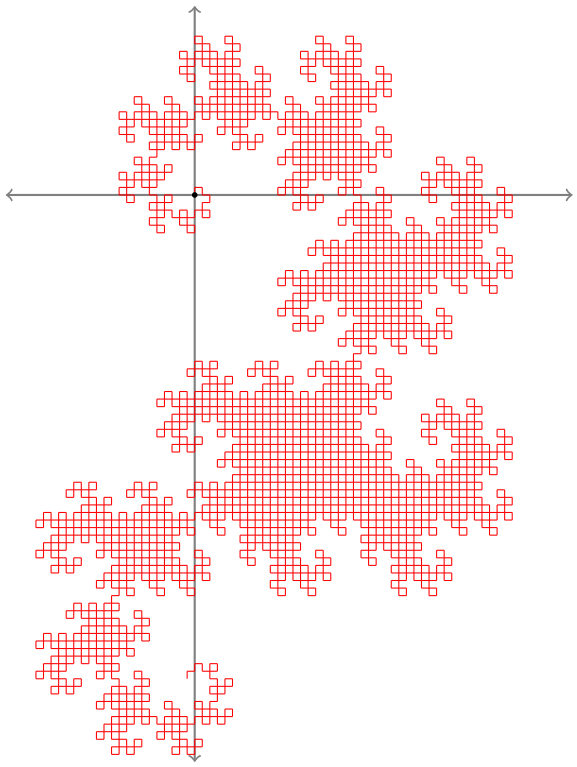
Entrada de ejemplo	**Salida de ejemplo**
5	2 -1

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

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

Entrada de ejemplo	**Salida de ejemplo**
4095	-1 -64

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



Primeros 4.095 pasos del camino del dragón alcanzando la coordenada (-1,-64)

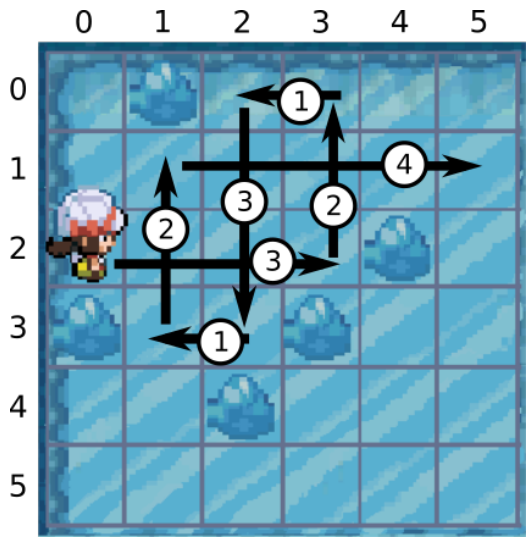
Problema
La cueva de hielo

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, éste 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 $(4, 2)$ 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)`
  - `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 caso contrario significa que la casilla 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.
  - `return`: La función debe retornar `true` si es posible para el laberinto especificado llevar al personaje desde la posición inicial a la final. En caso contrario debe retornar `false`.

Restricciones

- Para esta subtarea se probará la 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`.

Restricciones

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

Detalles de implementación

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

Grader de Ejemplo

Se provee un *grader* junto con 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.