

Dominó

13 de abril de 2016

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 cuadrado¹. 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

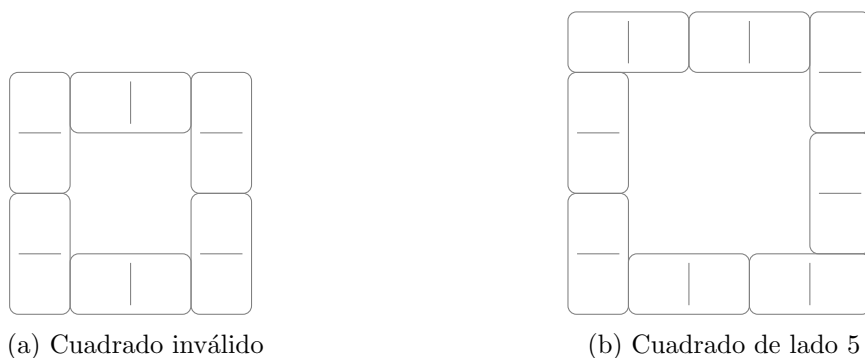


Figura 1: Un cuadrado válido y uno inválido.

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

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

(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 $(7, 90)$.

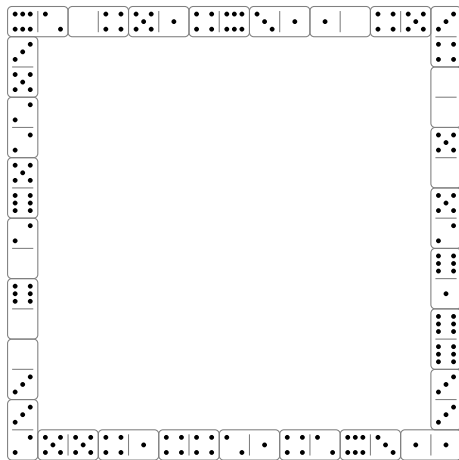


Figura 2: Un cuadrado $(7, 45)$. Notar que cada lado suma 45.

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.

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

Subtask 3 y 4

Para las subtasks 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 sea 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 subtask 3 se probará la función `validar` con varios casos donde $N \leq 17$ (20pts). Para la subtask 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