

Lenguaje de Programación

1

Recursividad, Divide y Vencerás, Backtracking

Bibliografía: “Estructura de Datos. Algoritmos, abstracción y objetos”.
Aguilar y Martínez. McGraw Hill 1998. Capítulo 9.

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, cubo(numero));
    }
    return 0;
}
```

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, cubo(numero));
    }
    return 0;
}
```

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

numero = 1

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, cubo(numero));
    }
    return 0;
}
```

numero = 1

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

base = numero = 1
potencia

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, cubo(numero));
    }
    return 0;
}
```

numero = 1

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

base = numero = 1
potencia = 1

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, 1);
    }
    return 0;
}
```

numero = 1

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

base = numero = 1
potencia = 1

Recordando a las Funciones en C

```
#include <stdio.h>
int cubo(int base);
main()
{
    int numero;
    for(numero=1; numero<=5; numero++){
        printf("El cubo del número %d es
        %d\n", numero, 1);
    }
    return 0;
}
```

```
int cubo(int base)
{
    int potencia;
    potencia = base * base * base;
    return potencia;
}
```

numero = 1

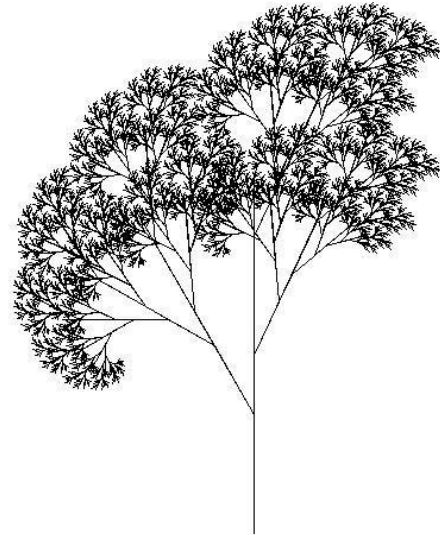
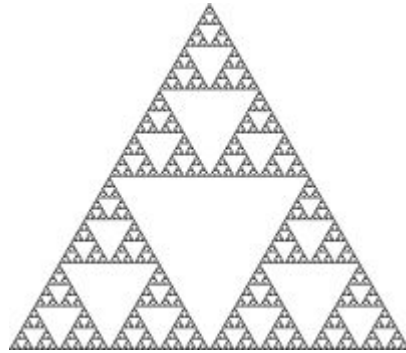
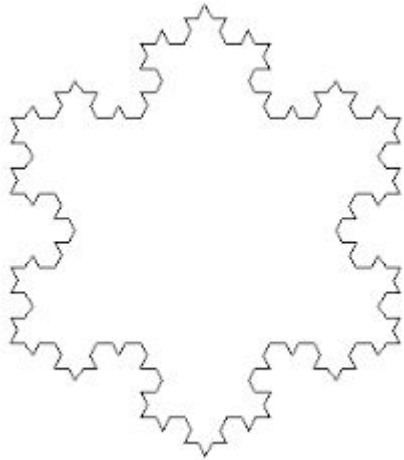
El cubo del número 1 es 1

Naturaleza de la Recursividad

- Los programas Estructurados se componen de funciones que se llaman unas a otras de manera organizada.
- Los programas RECURSIVOS se componen de Funciones que se llaman a sí mismas.
 - Necesita la definición del **CASO BASE** o condición de salida, que evitará que la función se llame a sí misma infinitamente.

Naturaleza de la Recursividad

- Hay muchos fenómenos recursivos:
 - Matemáticas: Fibonacci, Factorial, etc.
 - Realidad: Árboles, plantas, etc



Naturaleza de la Recursividad

- Hay muchos fenómenos recursivos:
 - Matemáticas: Fibonacci, Factorial, etc.
 - Realidad: Árboles, plantas, etc
- Ejemplo de Organización Recursiva:

Directa

```
void funcion1(...){  
    ...  
    funcion1();  
}
```

Indirecta

```
void funcion2(...){  
    ...  
    funcion3();  
}  
  
void funcion3(...){  
    ...  
    funcion2();  
}
```

Factorial

$$n! = n * (n-1) * (n-2) * \dots * 1$$

$$0! = 1; \quad 1! = 1; \quad 2! = 2 * 1; \quad 3! = 3 * 2 * 1$$

$$n! = n * (n-1)!$$

Factorial

$$n! = n * (n-1) * (n-2) * \dots * 1$$
$$0! = 1; \quad 1! = 1; \quad 2! = 2 * 1; \quad 3! = 3 * 2 * 1$$

$$n! = n * (n-1)!$$

$$(n-1)! = (n-1) * (n-2)!$$


Factorial

$$n! = n * (n-1) * (n-2) * \dots * 1$$

$$0! = 1; \quad 1! = 1; \quad 2! = 2 * 1; \quad 3! = 3 * 2 * 1$$

$$n! = n * (n-1)!$$

$$(n-1)! = (n-1) * (n-2)!$$


$$(n-k)! = (n-k) * (n-k-1)!$$

Factorial

$$n! = n * (n-1) * (n-2) * \dots * 1$$

$0! = 1; \quad 1! = 1; \quad 2! = 2 * 1; \quad 3! = 3 * 2 * 1$

```
long factorial(int n)
{
    long f=1;
    int i=n;
    While (i>0)
    {
        f = f*i;
        i = i-1;
    }
    return f;
}
```

Iterativamente

```
long factorial(int n)
{
    if (n==0) return 1;
    else
        return n *
        factorial(n-1);
}
```

Rekursivamente

RESOLVER Ejemplos

1. Deducir la definición recursiva del producto de números naturales usando solamente la operación SUMA.

2. Definir la serie de Fibonacci de manera recursiva e iterativa.

- Serie Fibonacci: 0,1,1,2,3,5,8....
- $F(n)=F(n-1)+F(n-2)$

(*)Indicar Eficiencia $F(N)$ y Complejidad Espacial

Respuesta 1

Condición de Salida: Cuando $b=1$ el producto de $a*b=a$;

```
int prod(int a, int b){  
    if (b==1)  
        return a;  
    else  
        return a+prod(a, b-1);  
}
```

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int});$

```
int prod(int a, int b){ //b=b-1  
    if (b==1)  
        return a;  
    else  
        return a+prod(a, b-1);  
}
```

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int});$

```
int prod(int a, int b){ //b=b-2  
    if (b==1)  
        return a;  
    else  
        return a+prod(a, b-1);  
}
```

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int});$

```
int prod(int a, int b){ //b=1  
    if (b==1)  
        return a;  
    else  
        return a+prod(a, b-1);  
}
```

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int});$

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int}) * \mathbf{b};$

Respuesta 1

Condición de Salida: Cuando $b=1$ el producto de $a*b=a$;

```
int prod(int a, int b) {  
    if (b==1)  
        return a;  
    else  
        return a+prod(a, b-1);  
}
```

$C_{\text{esp}} = 3 * \text{sizeof}(\text{int}) * b$; {se invoca b veces y se deben almacenar dos variables (a y b) en cada invocación}

$F(n) \cong (b-1)$

Respuesta 2

Recursivamente

```
long fibo(long n){  
    if (n==0 || n==1){  
        return n;  
    }else{  
        return fibo(n-2)+ fibo(n-1);  
    }  
}
```

$C_{\text{esp}} = \text{sizeof}(\text{long}) * (n-1);$
 $F(n) < 2^n$

Iterativamente

```
long fibo(long n){  
    long inf, sup, x;  
    if(n<=1){  
        return n;  
    }else{  
        inf=0; sup=1;  
        for(i=2; i<=n; i++){  
            x = inf;  
            inf = sup;  
            sup = x + inf;  
        }  
    }  
    return sup;  
}
```

$C_{\text{esp}} = \text{sizeof}(\text{long}) * 4$
 $F(n) = n$

Recursión Versus Iteración

	Iteración	Recursión
Estructura de Control	Repetitiva	Selectiva
Usa Estructura de Repetición?	La usa explícitamente	No, pero usa el stack del SO.
Cuando termina la ejecución?	Cuando la condición del bucle no se cumple	Cuando se reconoce el caso base
Desventajas	Los problemas con Naturaleza Recursiva no quedan claros con esta solución	Cada llamada a función requiere tiempo extra de procesador y más memoria pues se vuelven a copiar las variables al stack.
Condición crítica de tiempo y Memoria	Se usa esta técnica	No se la puede usar

¿Iteración o Recursión?

Se deberá usar una Solución Recursiva Si:

- ❑ No existe una solución Iterativa Sencilla y clara.
- ❑ O si los recursos del Sistema y los Tiempos de proceso no son críticos.

Una función con llamada recursiva como última sentencia (recursión final) se puede transformar en Iterativa reemplazando la llamada con un bucle cuya condición chequea el caso base.

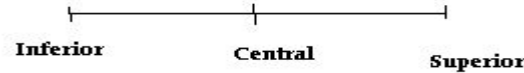
Algoritmos *Divide y Vencerás*

- Consiste en transformar un gran problema de tamaño n , en un conjunto de problemas pequeños, menores a n , pero similares.
 - Resolviendo los sub-problemas y combinando la soluciones, se resuelve el problema completo.
- Podría ser definido de manera recursiva llamándose a sí mismo con un conjunto menor de datos.
- Algunos ejemplos típicos de esta clase de Algoritmos son la Búsqueda y la Ordenación.

Búsqueda Binaria Recursiva

(con la técnica Divide y Vencerás)

Método de localización de una clave dentro de una arrays ordenado de n elementos.

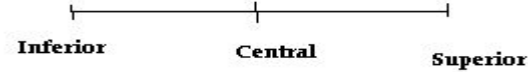


- Se tiene una lista ordenada $a[]$ con límite superior e inferior.
- Dada una clave se comienza la búsqueda en el Centro de la Lista:
 - $central = (inferior + superior) / 2$
 - Comparar $a[central]$ con clave
- Si $clave < a[central]$: Buscar entre inferior y $central - 1$
- Si $clave > a[central]$: Buscar entre $central + 1$ y superior

Búsqueda Binaria Recursiva

(con la técnica Divide y Vencerás)

Método de localización de una clave dentro de una arrays ordenado de n elementos.

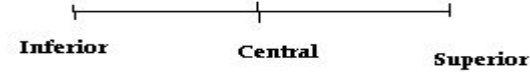


- Se tiene una lista ordenada $a[]$ con límite superior e inferior.
- Dada una clave se comienza la búsqueda en el Centro de la Lista:
 - $\text{central} = (\text{inferior} + \text{superior}) / 2$
 - Comparar $a[\text{central}]$ con clave
- Si $\text{clave} < a[\text{central}]$: Buscar entre inferior y $\text{central} - 1$
- Si $\text{clave} > a[\text{central}]$: Buscar entre $\text{central} + 1$ y superior

Búsqueda Binaria Recursiva

(con la técnica Divide y Vencerás)

Método de localización de una clave dentro de una arrays ordenado de n elementos.



- Se tiene una lista ordenada $a[]$ con límite superior e inferior.
- Dada una clave se comienza la búsqueda en el Centro de la Lista:
 - $\text{central} = (\text{inferior} + \text{superior}) / 2$
 - Comparar $a[\text{central}]$ con clave
- Si $\text{clave} < a[\text{central}]$: Buscar entre inferior y $\text{central}-1$

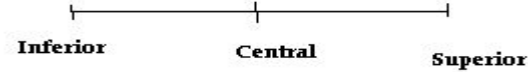


- Si $\text{clave} > a[\text{central}]$: Buscar entre $\text{central}+1$ y superior

Búsqueda Binaria Recursiva

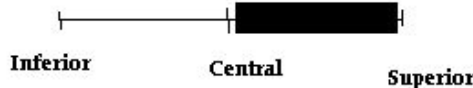
(con la técnica Divide y Vencerás)

Método de localización de una clave dentro de un array ordenado de n elementos.



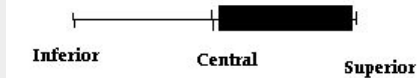
- Se tiene una lista ordenada $a[]$ con límite superior e inferior.
- Dada una clave se comienza la búsqueda en el Centro de la Lista:
 - $\text{central} = (\text{inferior} + \text{superior}) / 2$
 - Comparar $a[\text{central}]$ con clave
- Si $\text{clave} < a[\text{central}]$: Buscar entre inferior y $\text{central}-1$

- Si $\text{clave} > a[\text{central}]$: Buscar entre $\text{central}+1$ y superior



Búsqueda Binaria Recursiva (implementación)

```
int busquedaBR(int inferior, int superior, int clave, int V[]){  
    int central;  
    if (inferior>superior){ return -1; // No encontrado  
    }else{  
        central=(inferior+superior)/2;  
        if(V[central]==clave){ //Encontrado  
            return central;  
        }else if (V[central]<clave){  
            return busquedaBR(central+1, superior, clave,  
        }else{  
            return busquedaBR(inferior, central-1, clave  
        }  
    }  
}
```



BackTracking: Algoritmos de Vuelta Atrás

- El proceso general de estos algoritmos se basa en la Prueba y Búsqueda Exhaustiva.
 - Se Prueban todas las posibilidades (de una lista de posibles) Buscando la solución correcta.
 - Si una Posible Solución no conduce a la Solución del Problema, se vuelve atrás y se prueba con otra solución candidata de la lista.
- Estos algoritmos descomponen el proceso de Búsqueda en tareas parciales (Divide y Vencerás)
 - Se suele realizar en forma recursiva.

BackTracking: Esquema General

Función **ensayarsolucion**

Inicio

 Inicializar variables;

 Repetir

 Tomar Tarea[i] //de la lista de Tareas posibles

 Validar Tarea[i]; //Como posible solución

 Si TareaValida Entonces

 Anotar “i” en Solución[] //lista de Soluciones posibles

 Si ProblemaSolucionado Entonces

 exito=Verdadero

 Sino

ensayarsolucion //verifica si es o no una solución

 Si NO éxito => Borrar “i” de Solucion []

 //para probar con otra selección

 FinSi

 FinSi

 FinSi

 Hasta (exito o NoMasTareas)

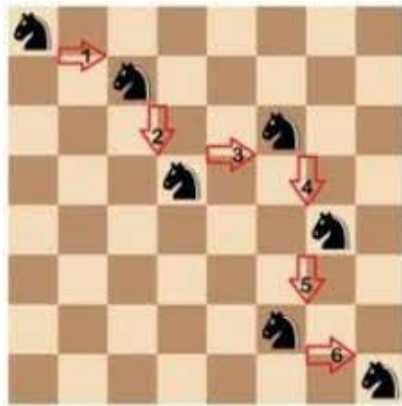
FIN Subprograma

El problema del Salto de Caballo



Problema del Salto de Caballo

- ❑ **Encontrar, si existe, un circuito que permita al caballo pasar una vez por cada casillero del tablero.**
- ❑ Un tablero de Ajedrez con $n \times n$ casillas, un caballo situado en la posición inicial de coordenadas (x_0, y_0) .



1	16	11	6	3
10	5	2	17	12
15	22	19	4	7
20	9	24	13	18
23	14	21	8	25

Problema del Salto de Caballo

- ▣ **Encontrar, si existe, un circuito que permita al caballo pasar una vez por cada casillero del tablero.**
- ▣ Un tablero de Ajedrez con $n \times n$ casillas, un caballo situado en la posición inicial de coordenadas (x_0, y_0) .
- ▣ Desde una casilla un caballo puede dar hasta 8 movimientos:
Número Posible de Selecciones = 8
 - $d = \{(2,1), (1,2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -2), (2, -1)\}$

	3		2	
4				1
		Θ		
5				8
	6		7	

Problema del Salto de Caballo

- ❑ **Encontrar, si existe, un circuito que permita al caballo pasar una vez por cada casillero del tablero.**
- ❑ Un tablero de Ajedrez con $n \times n$ casillas, un caballo situado en la posición inicial de coordenadas (x_0, y_0) .

The image displays four 8x8 chessboards illustrating knight tour solutions. The first three boards show partial tours, and the fourth board shows a complete tour with numbers 1-64.

1	12	17	22	3			
18	25	2	11	16			
13	8	23	4	21			
24	19	6	15	10			
7	14	9	20	5			

1	22	9	34	3	24		
10	33	2	23	16	31		
21	8	35	32	25	4		
36	11	26	17	30	15		
7	20	13	28	5	18		
12	27	6	19	14	29		

1	32	23	42	3	30	27	
22	47	2	31	28	43	4	
33	24	45	48	41	26	29	
46	21	34	25	44	5	38	
15	18	49	40	37	8	11	
20	35	16	13	10	39	6	
17	14	19	36	7	12	9	

1	16	39	22	3	18	49	56
38	23	2	17	60	55	4	19
15	40	63	54	21	48	57	50
24	37	42	59	64	61	20	5
41	14	53	62	47	58	51	30
36	25	46	43	52	31	6	9
13	44	27	34	11	8	29	32
26	35	12	45	28	33	10	7

Problema del Salto de Caballo

- ▣ **Encontrar, si existe, un circuito que permita al caballo pasar una vez por cada casillero del tablero.**
- ▣ Un tablero de Ajedrez con $n \times n$ casillas, un caballo situado en la posición inicial de coordenadas (x_0, y_0) .
- ▣ Desde una casilla un caballo puede dar hasta 8 movimientos:
Número Posible de Selecciones = 8
 - $d = \{(2,1), (1,2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -2), (2, -1)\}$
- ▣ Si la posición del caballo $(x, y) = (3, 5)$. PosibleSelecciones
 - $d = \{(5,6), (4,7), (2, 7), (1, 6), (1, 4), (2, 3), (4, 3), (5, 4)\}$

Problema del Salto de Caballo

- ▣ **Solución Parcial:** Que el caballo realice un movimiento Lícito dentro de los 8 posibles movimientos.
- ▣ Si la casilla destino
 - no está dentro del tablero la movida *no es lícita*.
 - si el caballo ya pasó por allí *no es lícita*.
 - De ser posible: se anota el salto.
- ▣ Si el caballo realizó n^2 saltos se pone éxito=Verdadero.

•Representación de los Datos:

- Tablero[x,y]=0 : no pasó el caballo
- Tablero[x,y]=i : pasó el caballo en el salto i

		2	7	
3	8			1
		⊖		6
9	4			
			5	

Codificación del Algoritmo

Salto de Caballo 1/3

```
#include <stdio.h>

#define N 8      /*Ancho del tablero*/
#define n (N+1)

void saltocaballo(int i, int x, int y, int
    *éxito);

void escribeTablero();

int tablero[n][n];

int d[8][2]={ (2,1), (1,2), (-1,2),
    (-2,1), (-2,-1), (-1,-2), (1,-2), (2,-1) };
```

Codificación ... (cont 2/3)

```
int main()
{   int x,y, solución, i, j;
    do{           /*Valores iniciales del tablero*/
        printf("\n Coordenadas Iniciales del caballo");
        scanf("%d %d",&x, &y);
    } while ((x<1)|| (x>N)  || (y<1)|| (y>N));

    for(i=1;i<=N;i++){           /*Inicializamos Matriz Tablero*/
        for(j=1;j<=N;j++){
            tablero[i][j]=0;
        }
    }

    tablero[x][y]=1;           /*Posición inicial del caballo*/
    solucion=0;
    saltocaballo(2,x,y,&solución);
    if(solucion){
        printf("\n\t El Problema tiene solución:\n");
        escribetablero();
    } else{
        printf("\n\t No se encontró solución al Problema \n");
    }
    return 0;
}
```

Codificación ... (cont 3/3)

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &exitito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	0	0

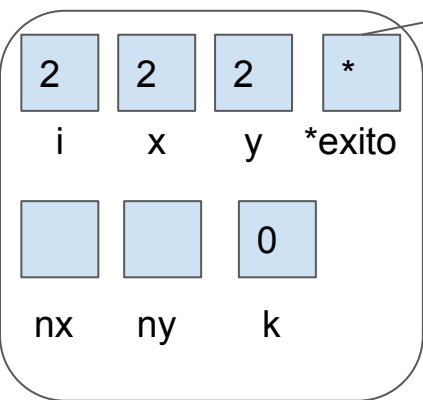
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	0	0

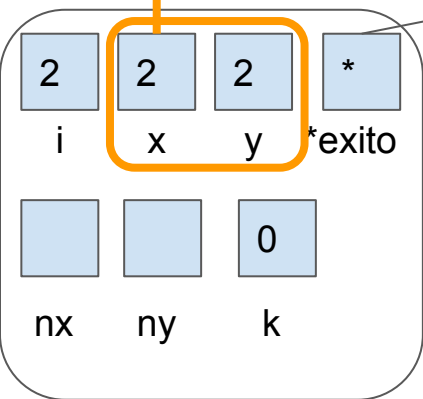
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	0	0

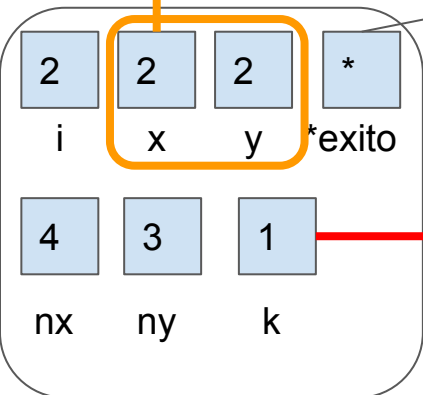
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	1	0	0	0
3	0	0	0	0
4	0	2	0	0

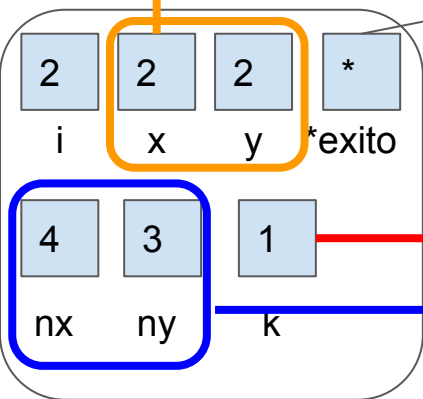
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

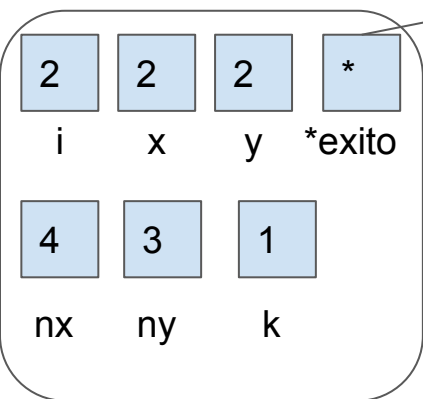
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

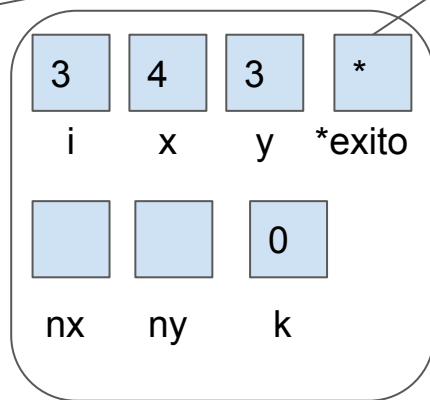
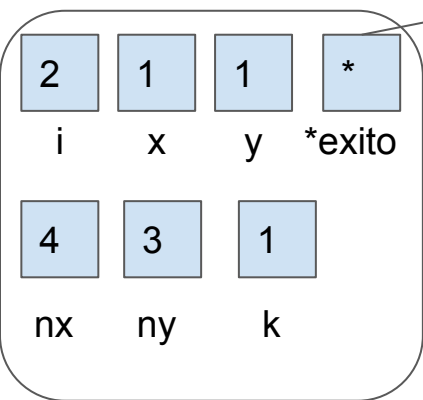
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

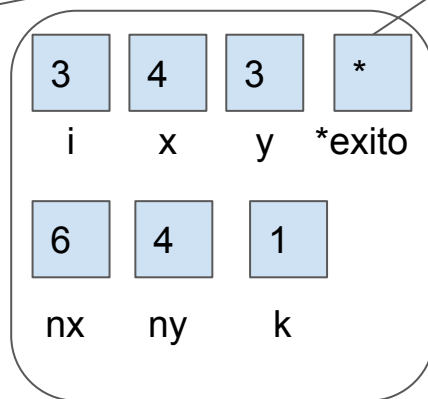
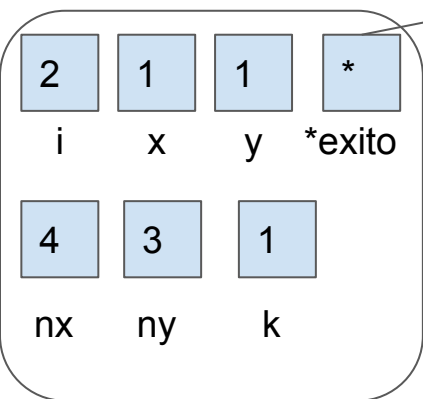
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

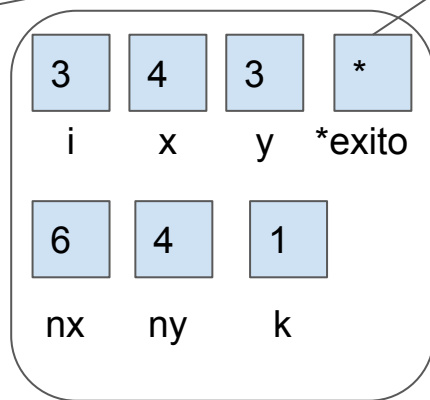
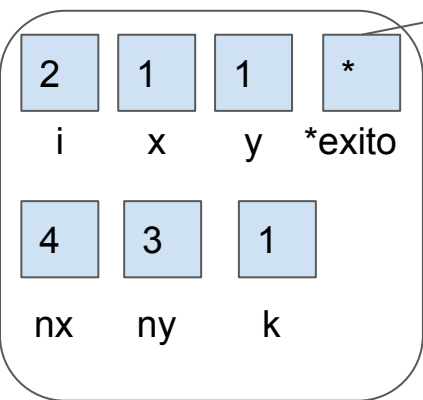
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

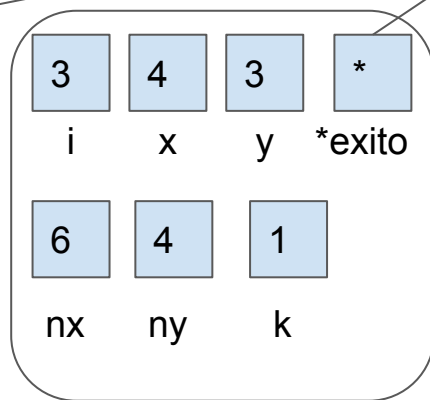
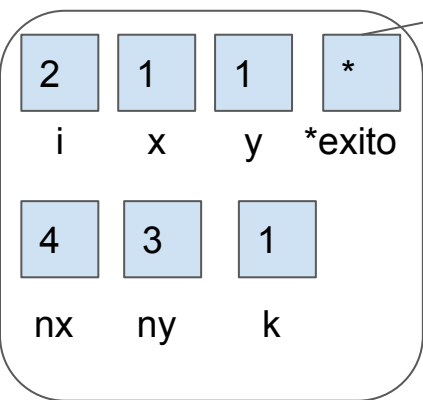
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

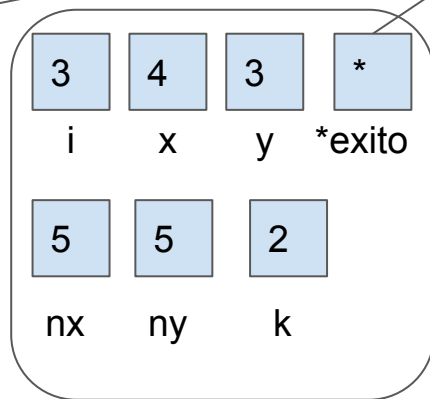
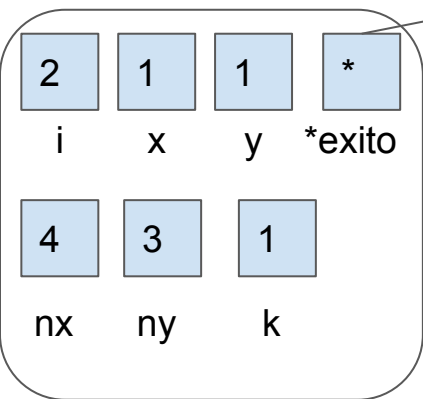
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

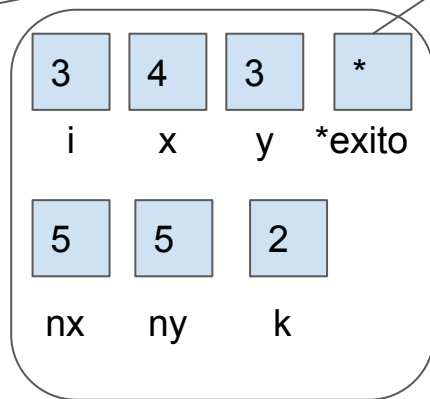
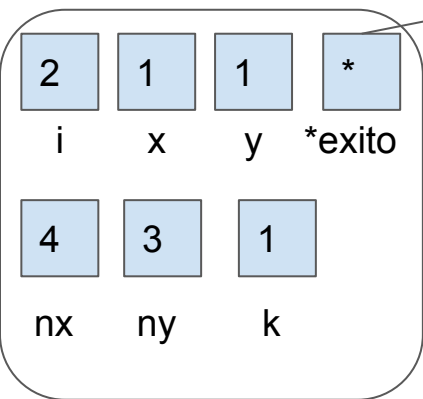
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

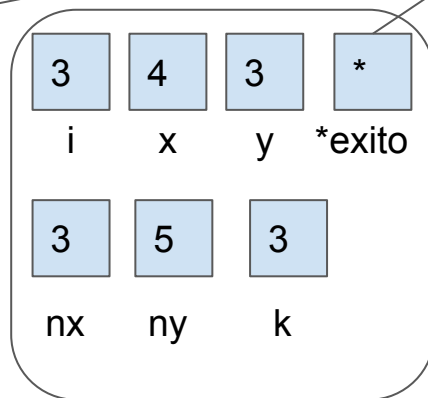
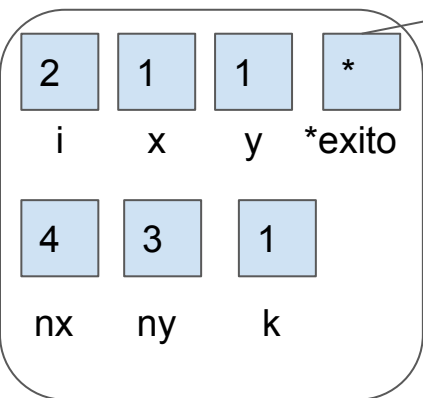
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	0
3	0	0	0	0
4	0	0	2	0

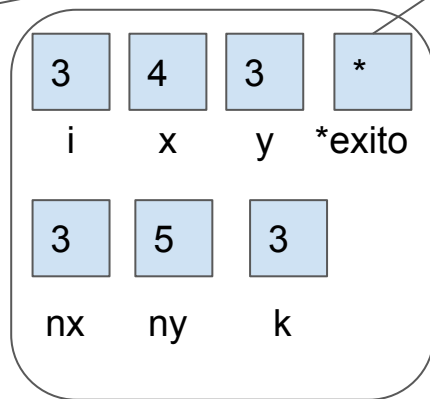
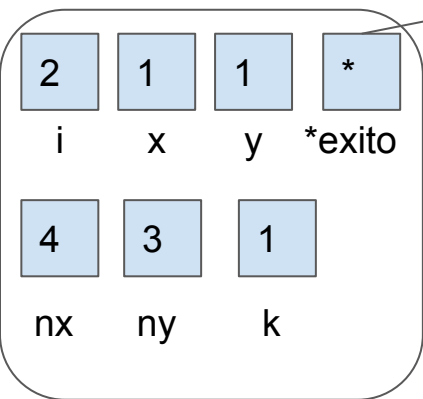
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

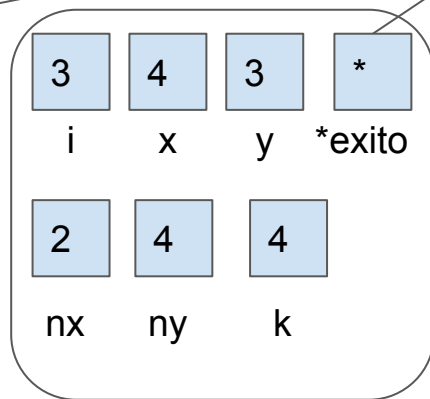
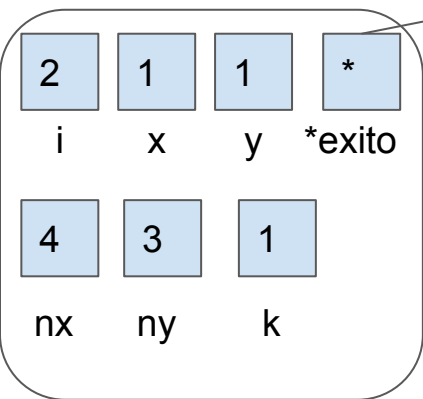
tablero

0

solucion

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

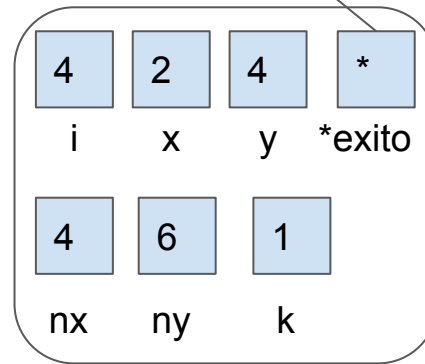
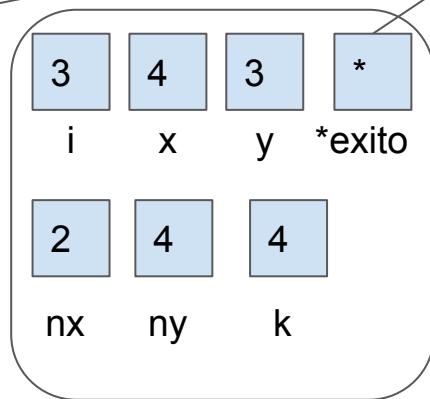
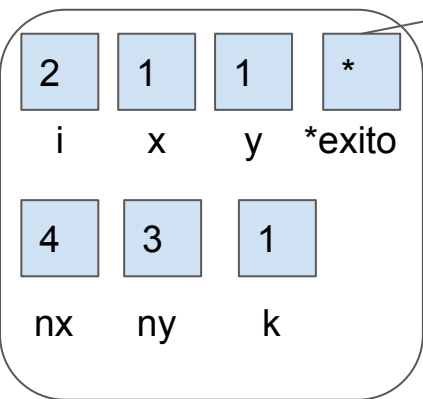
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

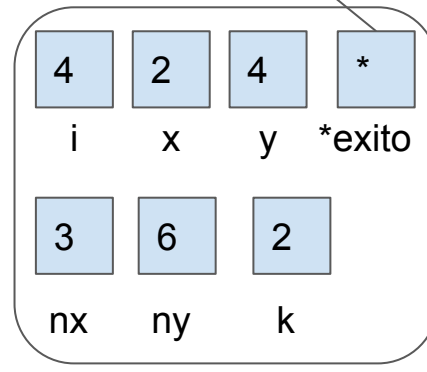
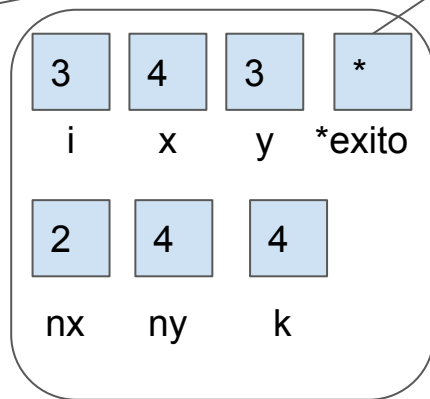
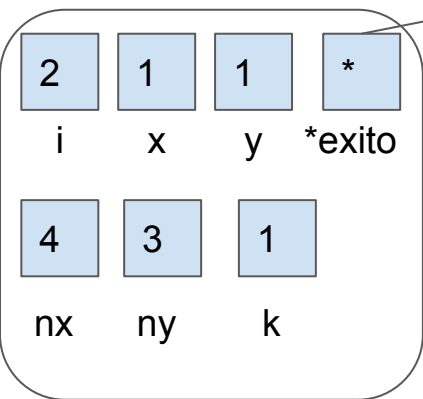
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

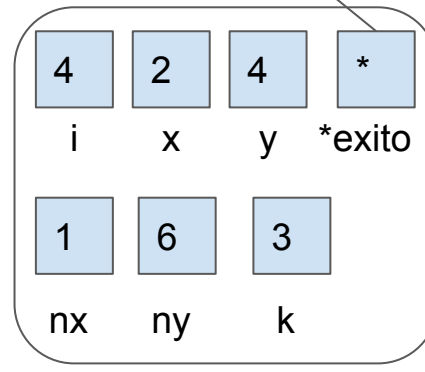
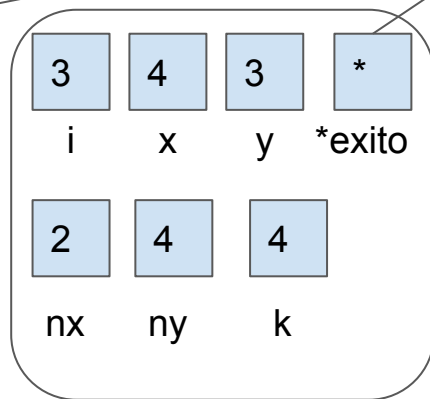
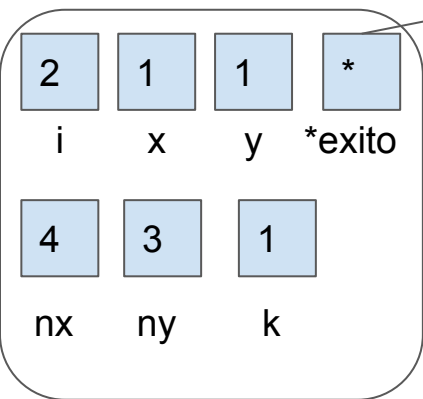
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

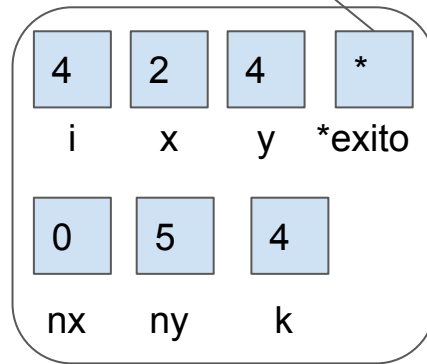
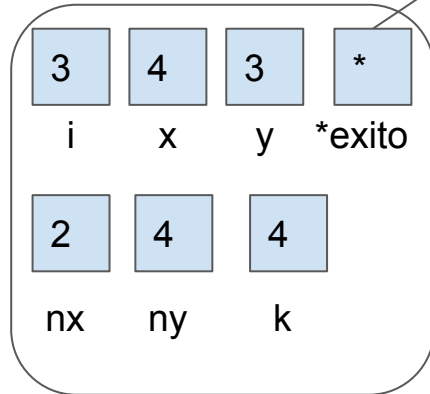
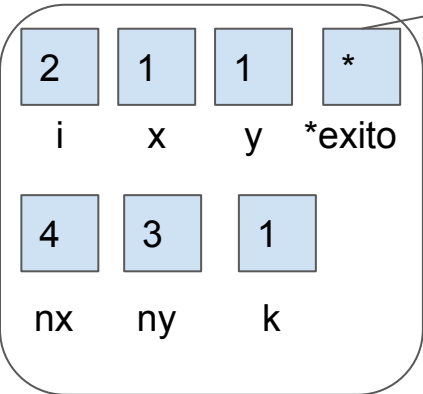
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	0	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

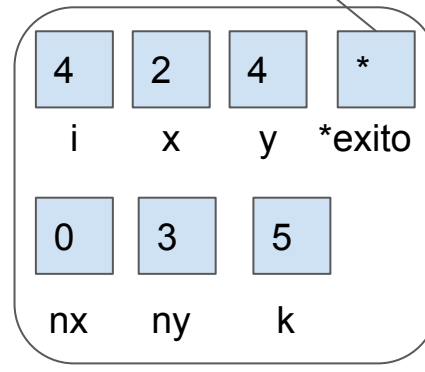
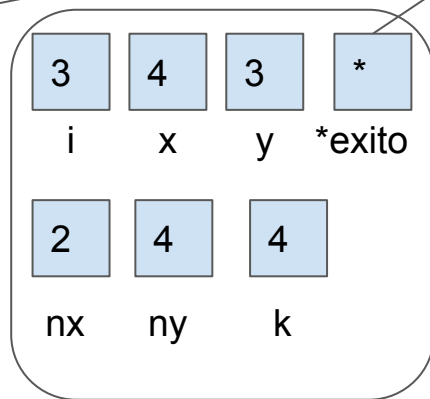
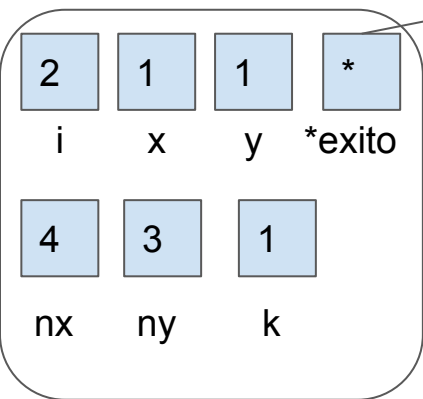
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	0	0
4	0	0	2	0

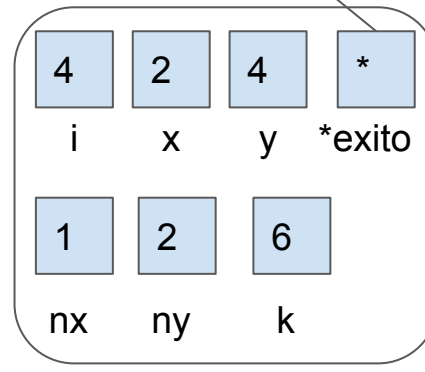
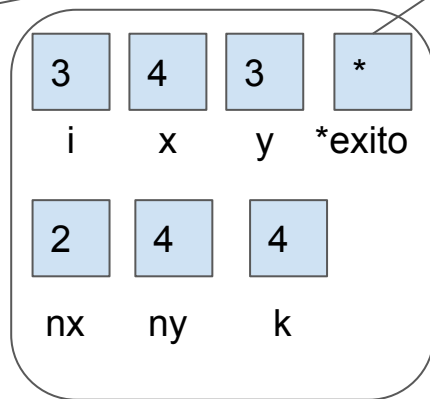
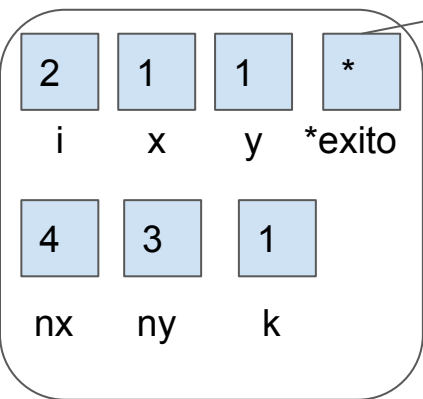
tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

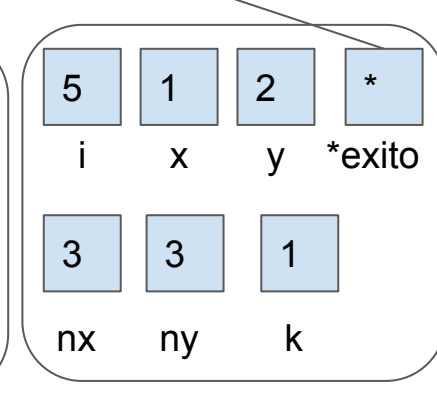
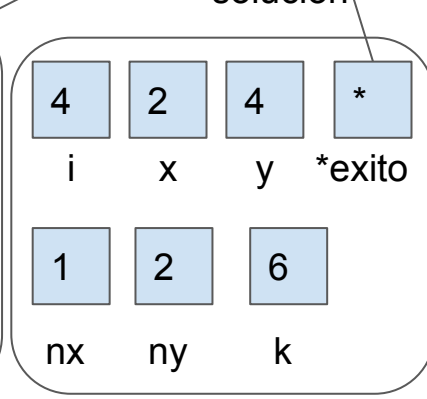
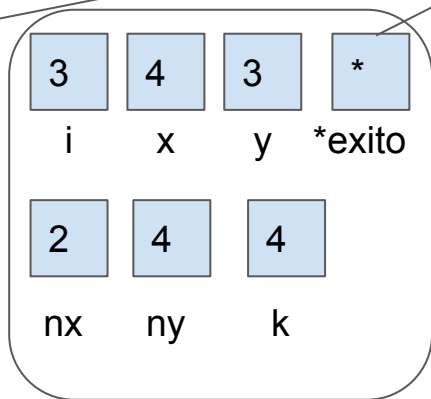
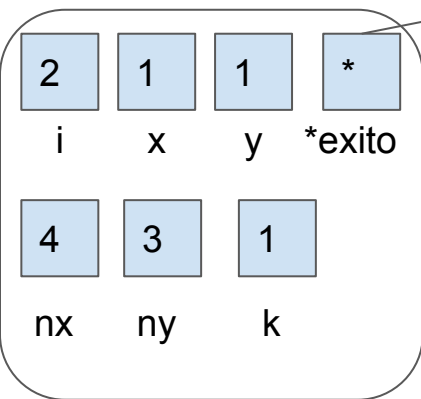
tablero

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

d



solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

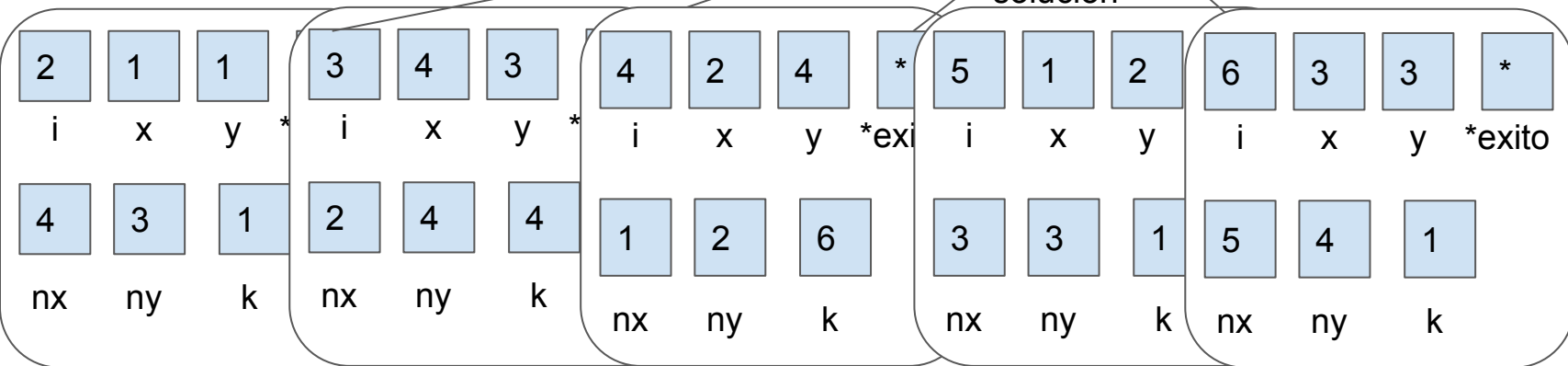
0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

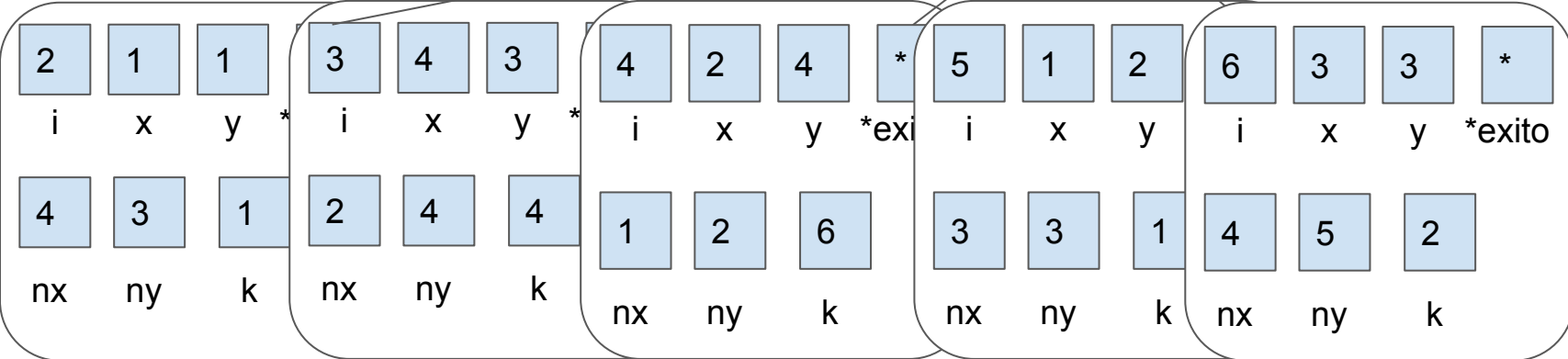
0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

d

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

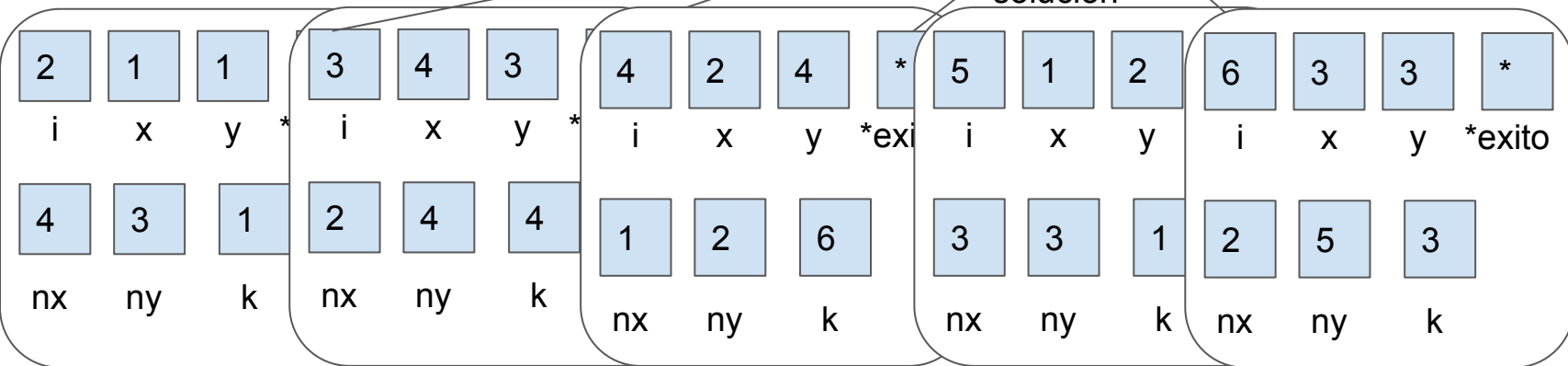
0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

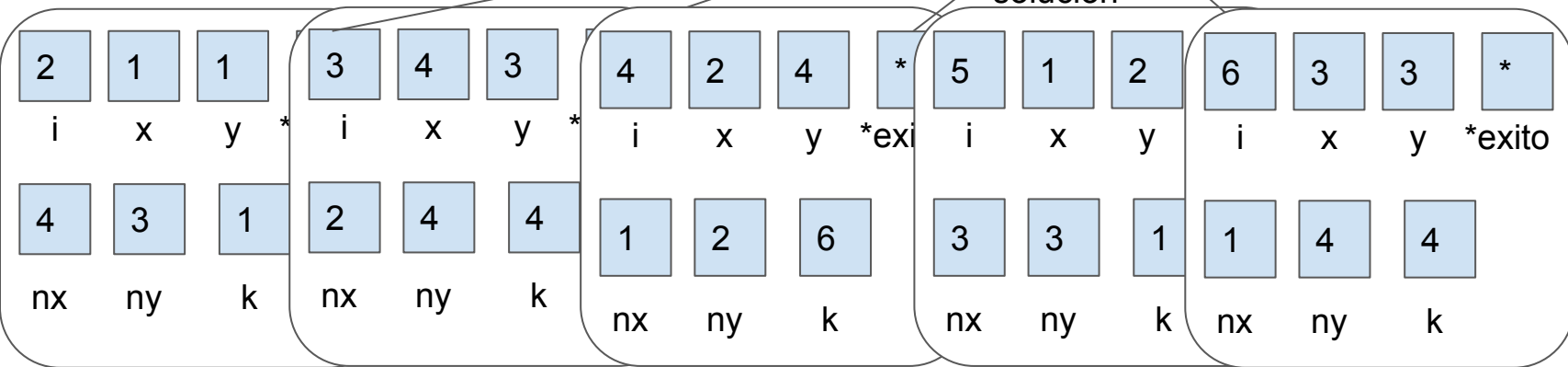
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) &&(ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

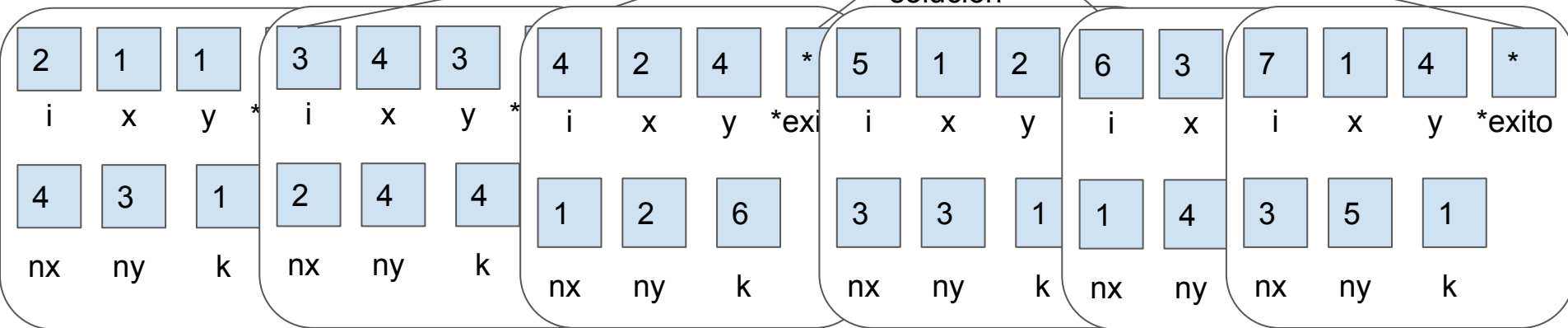
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

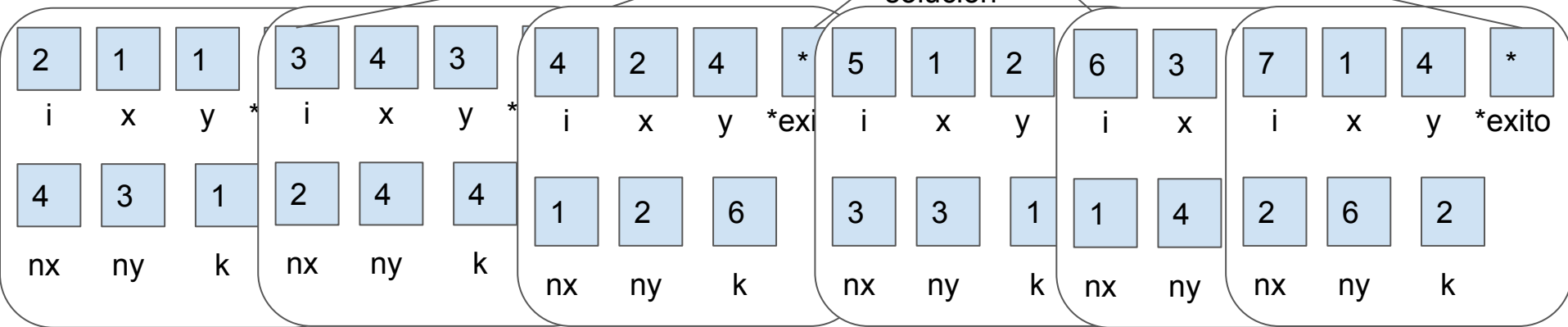
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

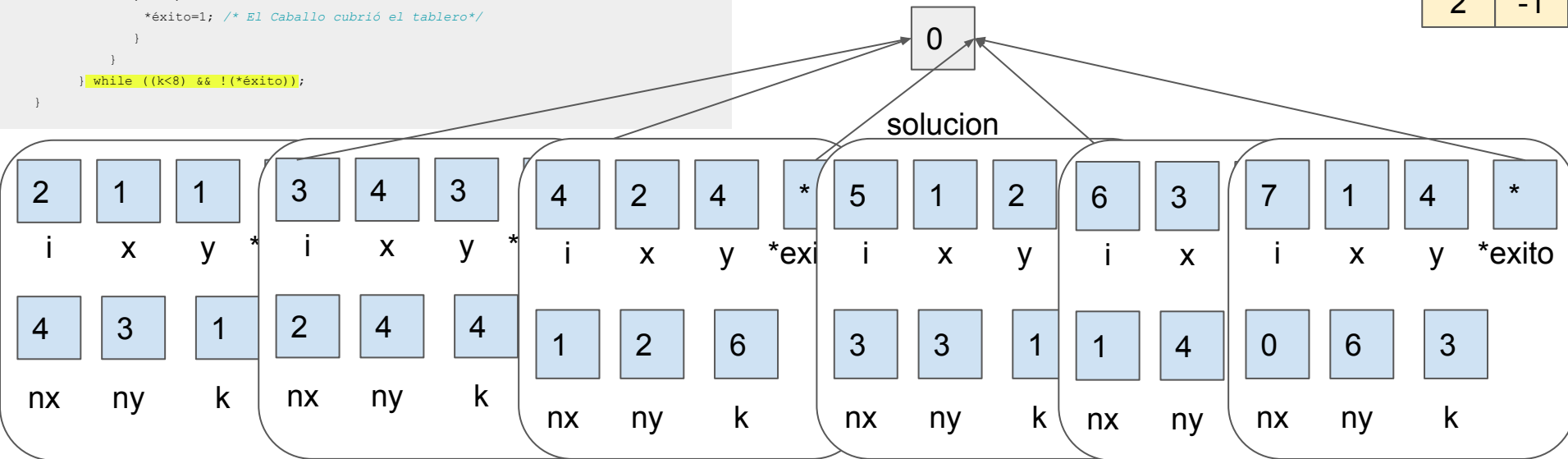
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

d



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

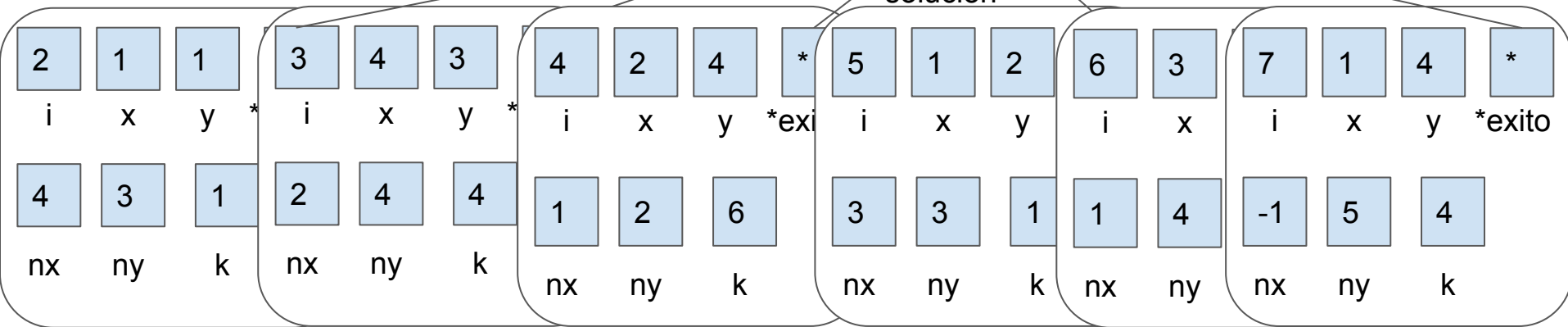
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

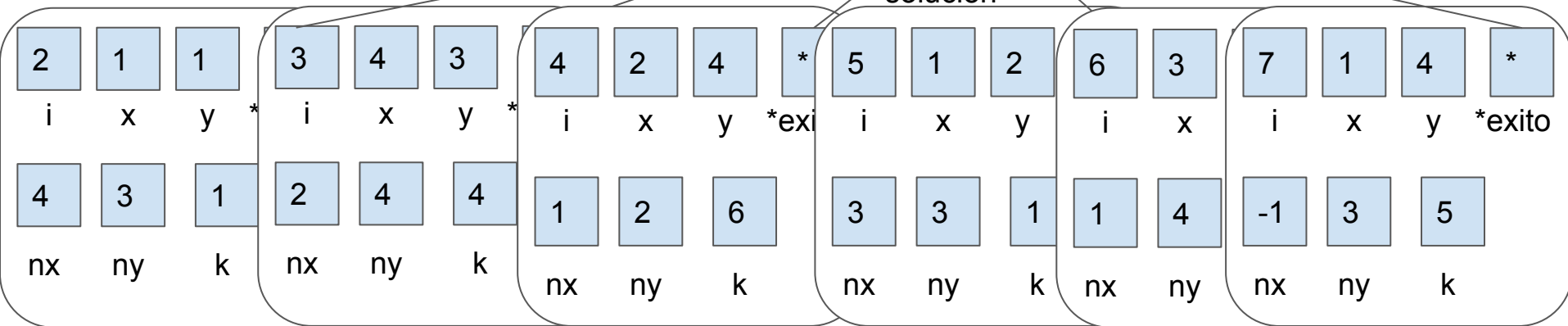
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

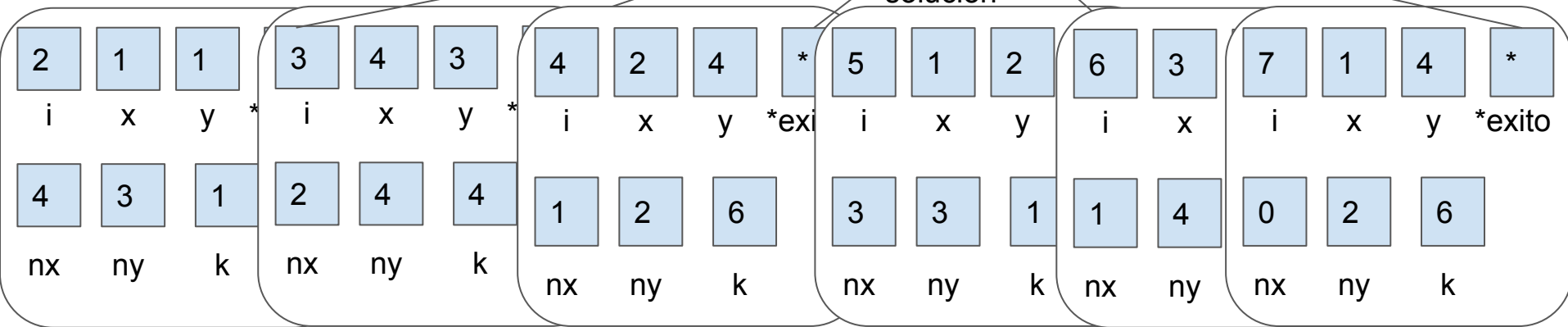
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

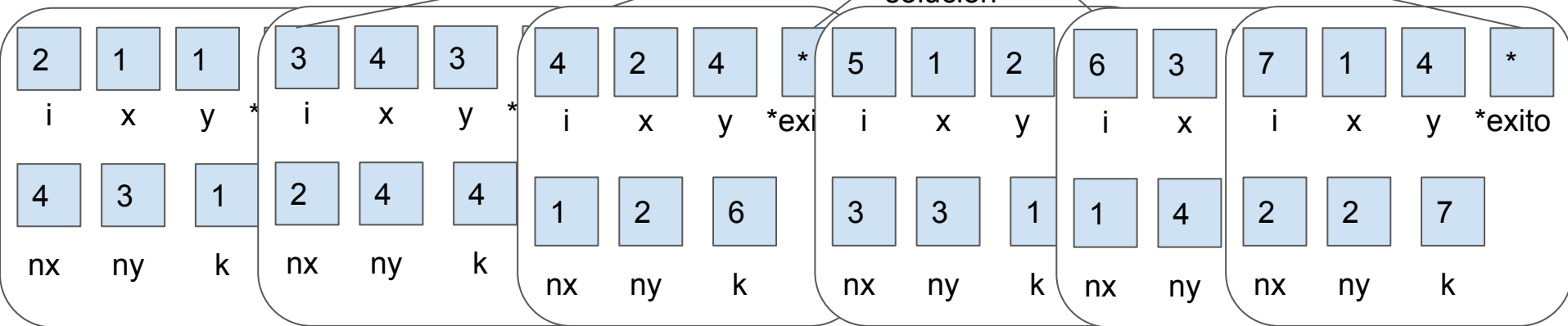
0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	4	0	6
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

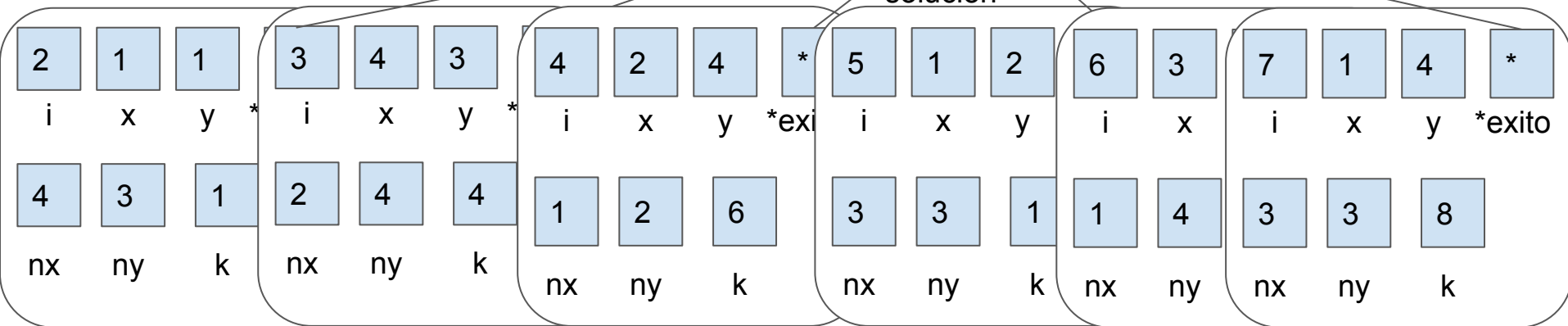
d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

0

solucion

Todos los movimientos fueron fallidos!!! **Backtracking**



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;
    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

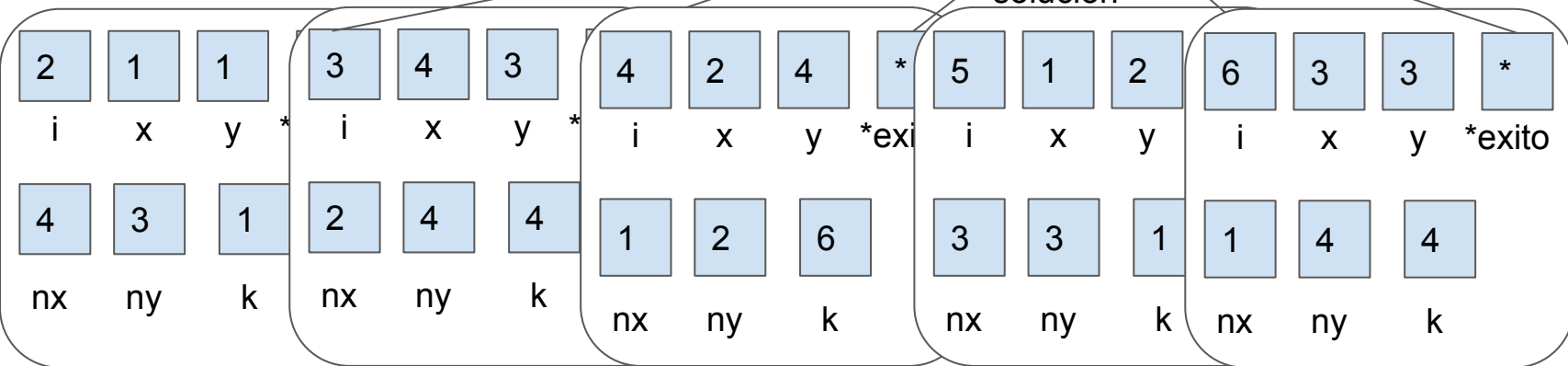
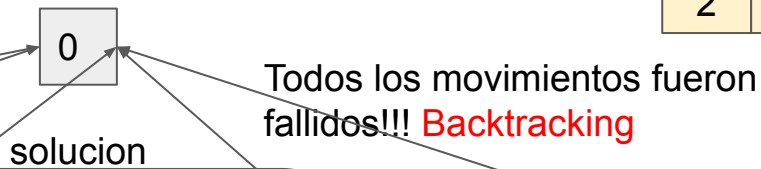
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

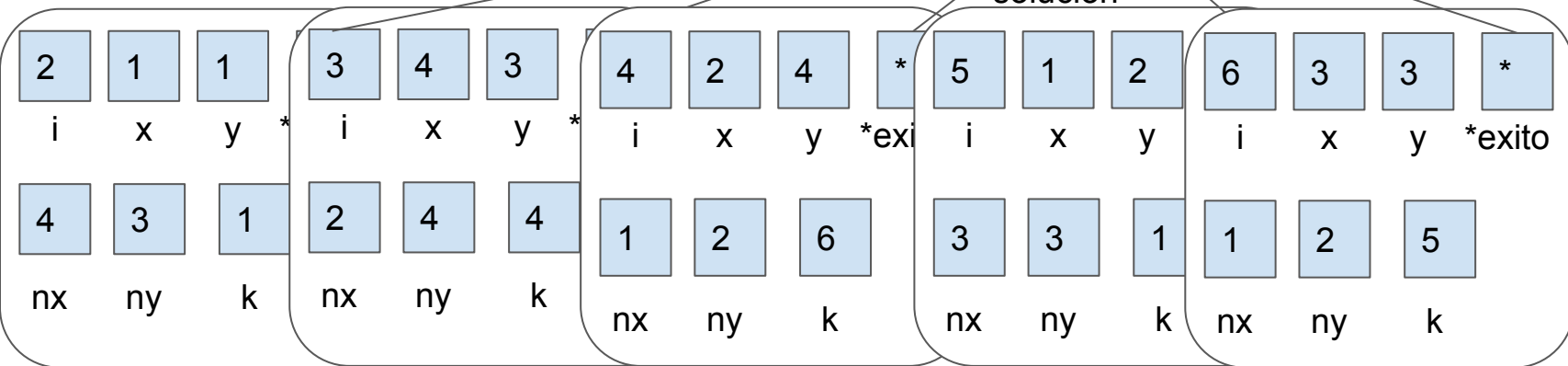
0	1	2	3	4
1	0	4	0	0
2	0	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/

        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if (!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

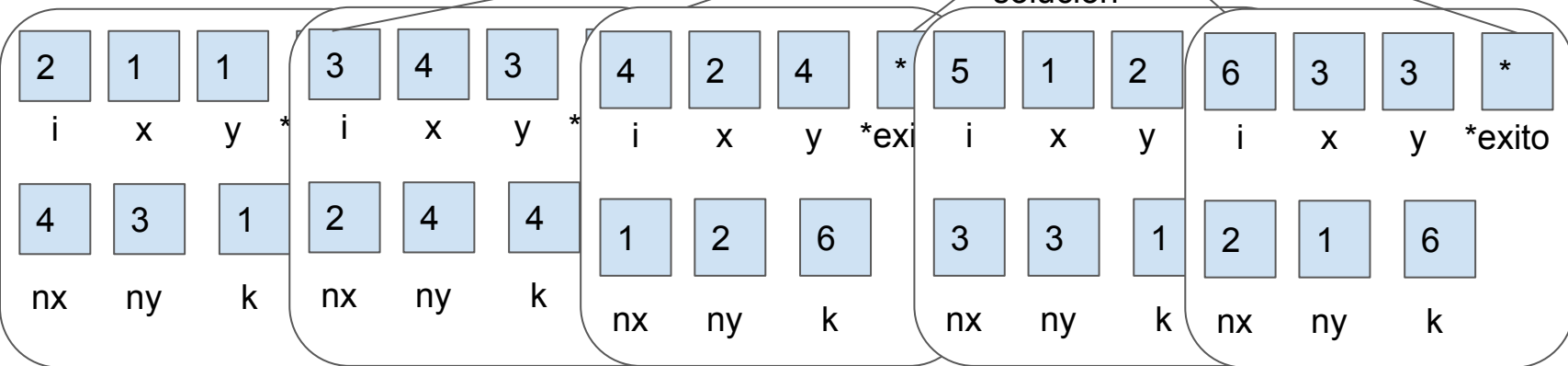
0	1	2	3	4
1	0	4	0	0
2	6	1	0	3
3	0	0	5	0
4	0	0	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Funcionamiento

```
void saltocaballo(int i, int x, int y, int *éxito){
    int nx,ny, k=0;

    *éxito=0;

    do {
        k++; /*Obtiene nuevas posiciones para el caballo*/
        nx = x + d[k-1][0];
        ny = y + d[k-1][1];
        /*determinar si estas posiciones son válidas y aún no se pasó por esa posición*/
        if ((nx>=1) && (nx <= N) && (ny >= 1) && (ny<=N) && tablero[nx][ny]==0){
            tablero[nx][ny] = i; /* Anotar movimiento*/
            if (i < N*N) {
                saltocaballo(i+1, nx, ny, &éxito);
                /* Analiza si se ha completado la solución*/
                if(!(*éxito)) tablero[nx][ny] = 0; /* Borra anotación*/
            }else{
                *éxito=1; /* El Caballo cubrió el tablero*/
            }
        }
    } while ((k<8) && !(*éxito));
}
```

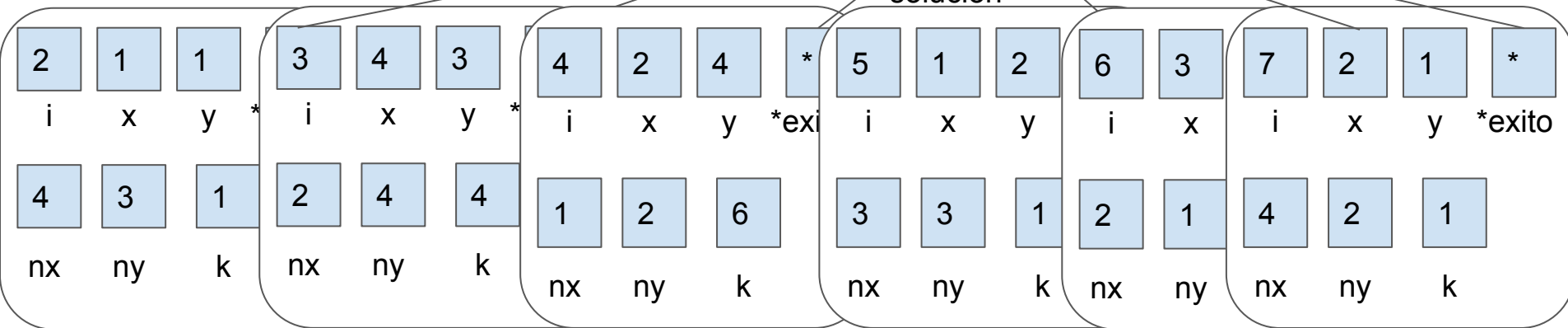
0	1	2	3	4
1	0	4	0	0
2	6	1	0	3
3	0	0	5	0
4	0	7	2	0

tablero

d

2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1

solucion



Lenguaje de Programación

1

Recursividad, Divide y Vencerás, Backtracking

Bibliografía: “Estructura de Datos. Algoritmos, abstracción y objetos”.
Aguilar y Martínez. McGraw Hill 1998. Capítulo 9.