Segundo Examen de Programación: Colonia de Hormigas

Curso 2024

Una colonia de hormigas se representa con una matriz de enteros en la que cada celda tiene un significado específico:

- El valor `0` representa un agujero (un espacio libre por el que las hormigas pueden transitar).
- Cualquier valor negativo representa tierra (por la que las hormigas no pueden transitar).
- Cualquier valor positivo representa el valor energético del alimento almacenado en esa cámara.

La entrada a la colonia siempre está en la posición `(0,0)`. El objetivo del problema es encontrar la secuencia de posiciones por las que debe transitar una hormiga (partiendo desde la entrada de la colonia) para recolectar cero o más alimentos y regresar a la entrada de forma que los alimentos que carga acumulen el mayor valor energético posible. Sin embargo, hay ciertas restricciones:

1. **Recoger Alimentos**: La hormiga **siempre** recoge alimentos mientras se mueve, o sea, si la hormiga transita por una cámara con un alimento, inevitablemente lo recogerá antes de seguir caminando. Sin embargo, si el valor energético total de los alimentos recolectados por la hormiga excede un umbral, la hormiga deja de recolectar más alimentos, incluso si hay más disponibles. Ojo que el alimento hay que cargarlo completo, no es posible

quedarse con solo una porción del alimento. Nótese además que la carga de la hormiga puede superar el umbral de carga si antes de cargar el último alimento todavía estaba por debajo del umbral.

2. **Movimiento**: La hormiga puede moverse en cuatro direcciones: arriba, abajo, izquierda y derecha. Además, no puede pasar a través de la tierra (*valores negativos*). La hormiga tampoco puede volver a pasar por una cámara por la que ya ha pasado, a menos que esté regresando a la entrada (en cuyo caso ya no recogerá más alimentos aunque sea posible).

Implemente el siguiente método en C# para resolver el problema:

```
public class AntColony
{
    // Método principal para encontrar la ruta óptima
    public int[] Solve(int[,] colony, int threshold)
    {
        // Escribe tu código aquí
    }
}
```

La longitud del array de salida debe coincidir con la longitud del camino tomado. Cada entero en el array codifica las posiciones por las que fue transitando la hormiga. La cámara `(i,j)` se codifica como `(i * número de columnas) + j`, o sea, las cámaras se numeran primero de izquierda a derecha y luego de arriba hacia abajo. Según esta codificación, el primer y último elemento del array resultante debe ser `0` (pues codifica la posición `(0,0)` de la colonia).

Escriba el código necesario que permita a la hormiga encontrar la ruta óptima para recolectar alimentos y regresar a la entrada, respetando las restricciones mencionadas anteriormente. La longitud del camino no necesariamente tendrá que ser la menor posible siempre que se satisfagan el resto de restricciones.

Aclaraciones

Puede asumir que:

- El array bidimensional `colony` no será `null`.
- El entero `threshold` será no negativo.

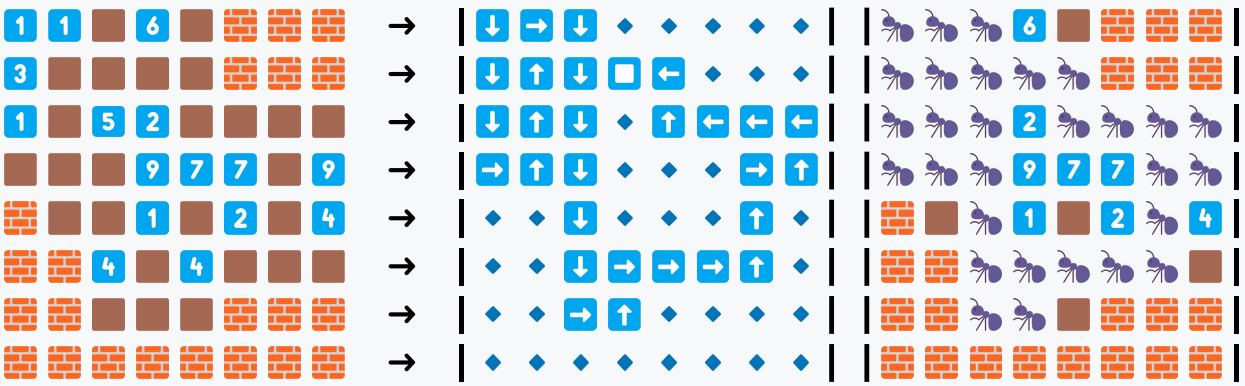
Ejemplo 1

```
int[,] colony = new int[,] {
    { 1 ,  1,   0,   6,   0,  -1,  -1,  -1 },
    { 3 ,  0,   0,   0,   0,  -1,  -1,  -1 },
    { 1 ,  0,   5,   2,   0,   0,   0,   0 },
    { 0 ,  0,   0,   9,   7,   7,   0,   9 },
    {-1,   0,   0,   1,   0,   2,   0,   4 },
    {-1,  -1,   4,   0,   4,   0,   0,   0 },
    {-1,  -1,   0,   0,   0,  -1,  -1,  -1 },
    {-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1 }
};
int[] solution = AntColony.Solve(colony, 20);
```

Una alternativa

```
// ENERGY
// 28
//
// SOLUTION
// { 0, 8, 16, 24, 25, 17, 9, 1, 2, 10, 18, 26, 34, 42, 50, 51, 43, 44,
45, 46, 38, 30, 31, 23, 22, 21, 20, 12, 11, // ida
//   12, 20, 21, 22, 23, 31, 30, 38, 46, 45, 44, 43, 51, 50, 42, 34, 26,
18, 10, 2, 1, 9, 17, 25, 24, 16, 8, 0 } // vuelta
```

Recorrido

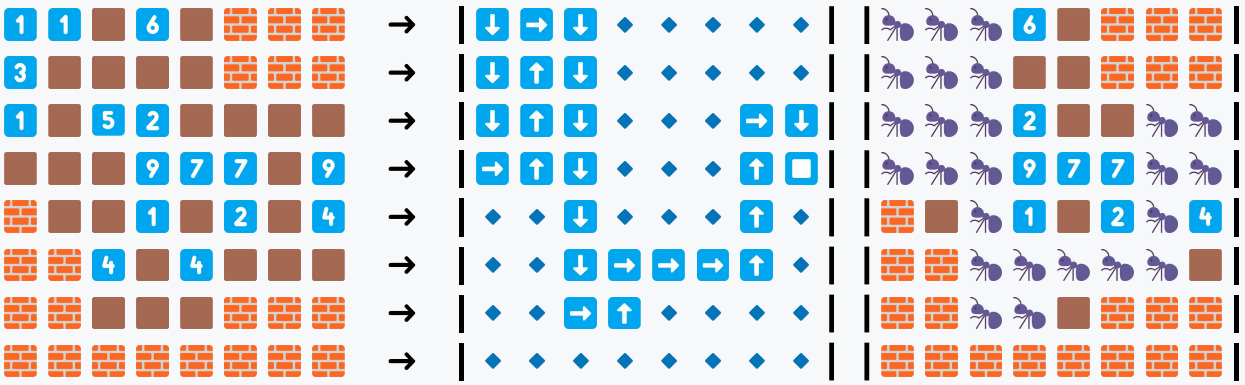


Total de energía recogida

Otra alternativa igual de válida

```
// ENERGY
// 28
//
// SOLUTION
// { 0, 8, 16, 24, 25, 17, 9, 1, 2, 10, 18, 26, 34, 42, 50, 51, 43, 44,
45, 46, 38, 30, 22, 23, 31, // ida
//   23, 22, 30, 38, 46, 45, 44, 43, 51, 50, 42, 34, 26, 18, 10, 2, 1, 9,
17, 25, 24, 16, 8, 0 } // vuelta
```

Recorrido



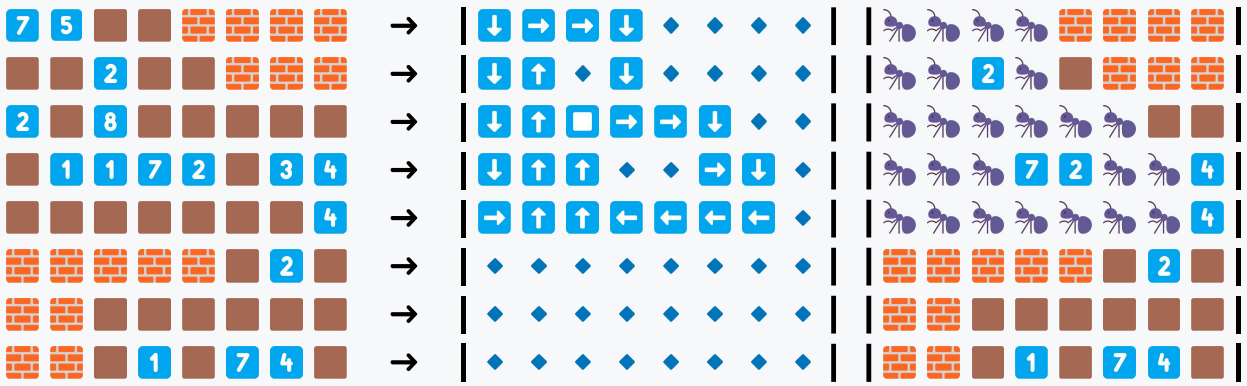
Total de energía recogida

1 3 1 1 5 4 4 9 = 28

Ejemplo 2

```
int[,] colony = new int[,] {
    {  7,   5,   0,   0,  -1,  -1,  -1,  -1 },
    {  0,   0,   2,   0,   0,  -1,  -1,  -1 },
    {  2,   0,   8,   0,   0,   0,   0,   0 },
    {  0,   1,   1,   7,   2,   0,   3,   4 },
    {  0,   0,   0,   0,   0,   0,   0,   4 },
    { -1,  -1,  -1,  -1,  -1,   0,   2,   0 },
    { -1,  -1,   0,   0,   0,   0,   0,   0 },
    { -1,  -1,   0,   1,   0,   7,   4,   0 }
};
int[] solution = AntColony.Solve(colony, 20);
//
// ENERGY
// 27
//
// SOLUTION
// { 0, 8, 16, 24, 32, 33, 25, 17, 9, 1, 2, 3, 11, 19, 20, 21, 29, 30,
38, 37, 36, 35, 34, 26, 18, // ida
//   26, 34, 35, 36, 37, 38, 30, 29, 21, 20, 19, 11, 3, 2, 1, 9, 17, 25,
33, 32, 24, 16, 8, 0 } // vuelta
```

Recorrido

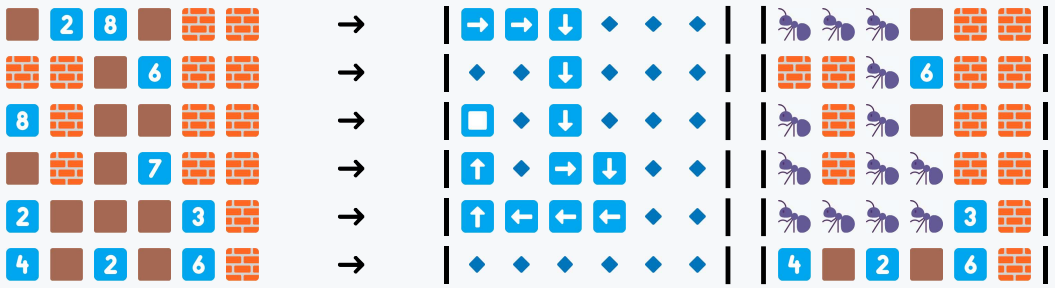


Total de energía recogida

Ejemplo 3

```
int[,] colony = new int[,] {
    {  0,   2,   8,   0,   -1,   -1 },
    { -1,  -1,   0,   6,   -1,   -1 },
    {  8,  -1,   0,   0,   -1,   -1 },
    {  0,  -1,   0,   7,   -1,   -1 },
    {  2,   0,   0,   0,    3,   -1 },
    {  4,   0,   2,   0,    6,   -1 }
};
int[] solution = AntColony.Solve(colony, 20);
//
// ENERGY
// 27
//
// SOLUTION
// { 0, 1, 2, 8, 14, 20, 21, 27, 26, 25, 24, 18, 12, // ida
//   18, 24, 25, 26, 27, 21, 20, 14, 8, 2, 1, 0 } // vuelta
```

Recorrido



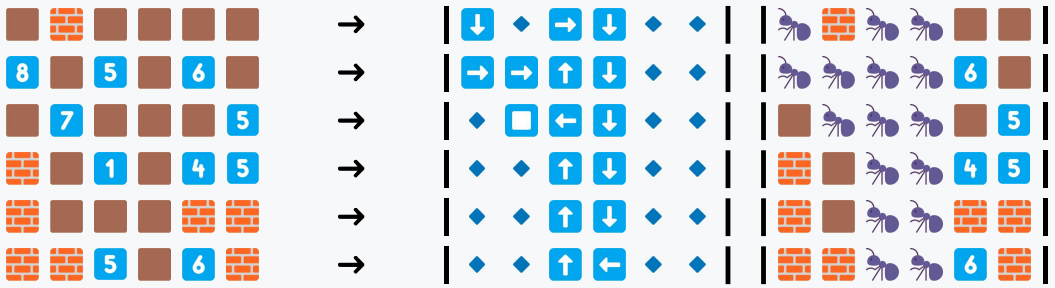
Total de energía recogida

2 8 7 2 8 = 27

Ejemplo 4

```
int[,] colony = new int[,] {
    {  0,  -1,    0,    0,    0,    0 },
    {  8,   0,    5,    0,    6,    0 },
    {  0,   7,    0,    0,    0,    5 },
    { -1,   0,    1,    0,    4,    5 },
    { -1,   0,    0,    0,   -1,   -1 },
    { -1,  -1,    5,    0,    6,   -1 }
};
int[] solution = AntColony.Solve(colony, 20);
//
// ENERGY
// 26
//
// SOLUTION
// { 0, 6, 7, 8, 2, 3, 9, 15, 21, 27, 33, 32, 26, 20, 14, 13, // ida
//   14, 20, 26, 32, 33, 27, 21, 15, 9, 3, 2, 8, 7, 6, 0 } // vuelta
```

Recorrido



Total de energía recogida

8 5 5 1 7 = 26