Realizado por: Samuel David Villegas Bedoya
Explicación de la solución.

Descripción del algoritmo:

→Primero ponemos los diferentes casos de la entrada en un Array, para luego encontrarles la solución.
→ Iteramos el Array y llamamos el método **process** el cual se encarga de crear las estructuras necesarias que utiliza el método recursivo (**auxProcess**) para funcionar y también hacemos la estructura de respuesta, es decir, mostramos en pantalla la estructura principal de la respuesta, así:

Layout #:
//Mostramos el layout o el test a validar

Maps resulting from layout # are:

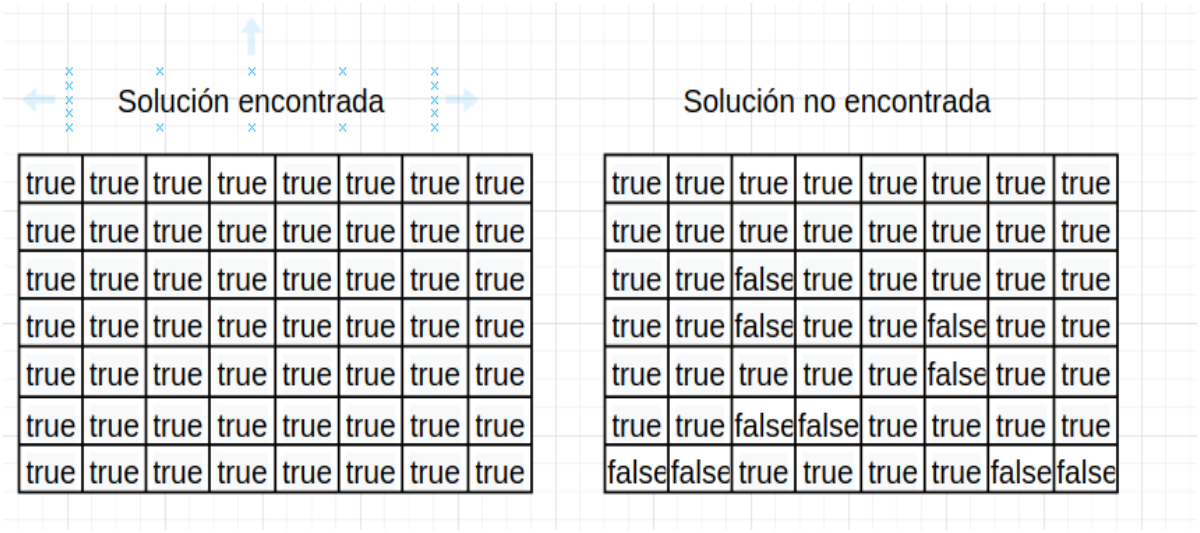
//Aqui se llama el método recursivo, el cual va a mostrar todos las soluciones que encuentre.

There are 2 solution(s) for layout #.

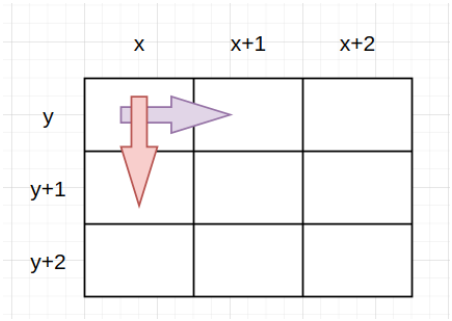
Cuando ya pasamos al método recursivo que es el que hace la mayoría del trabajo podemos describir diferentes estructuras utilizadas para su correcto funcionamiento, entre ellas tenemos:

- **test**: String[][] => Matriz de 7x8 donde guardamos el test a validar.
- **auxiliarMatriz**: String[][] => Matriz 7x8 donde ingresaremos la solución, es decir, utilizamos los **bones** para llenarlo a medida que recorremos el test.
- **VisitedFields**: boolean[][]=> Matriz 7x8 donde se guardará si hemos visitado o no un campo. En el caso de que sea verdadero ese campo no se puede tomar.
- **activatedBones**: boolean[]=>Arreglo de 28 posiciones donde guardamos si un hueso ya está utilizado o no. Si está utilizado no lo podemos tomar.
- **bones**: String[][]=>Matriz 7x7 donde guardamos el valor del hueso correspondiente a cada par de coordenadas. Ej: (0,0) => 1 (0,1)=>2 (1,0)=>2
- **row,col:** int =>Es el par de coordenadas del campo a visitar.
- **numberSolution**: int[] => Es un arreglo de una posición que contiene el numero de soluciones que ha tenido el layout, se ubicó en un arreglo ya que como es un objeto podemos modificarlo dentro del método y obtener el valor afuera, sin necesidad que el método recursivo me retorne el valor.

¿Por qué funciona? Y mecanismo de Backtraking:
El algoritmo cuenta con varias partes fundamentales para su buen funcionamiento:
0. Primero verificamos que halla o no una solución, para esto llamamos al método visited() el cual retorna true si todos los elementos están visitados, es decir, se encontró una solución, en ese caso se imprimen los valores de la matriz **auxiliarMatriz,** de ser falso se ingresa en el proceso de búsqueda.



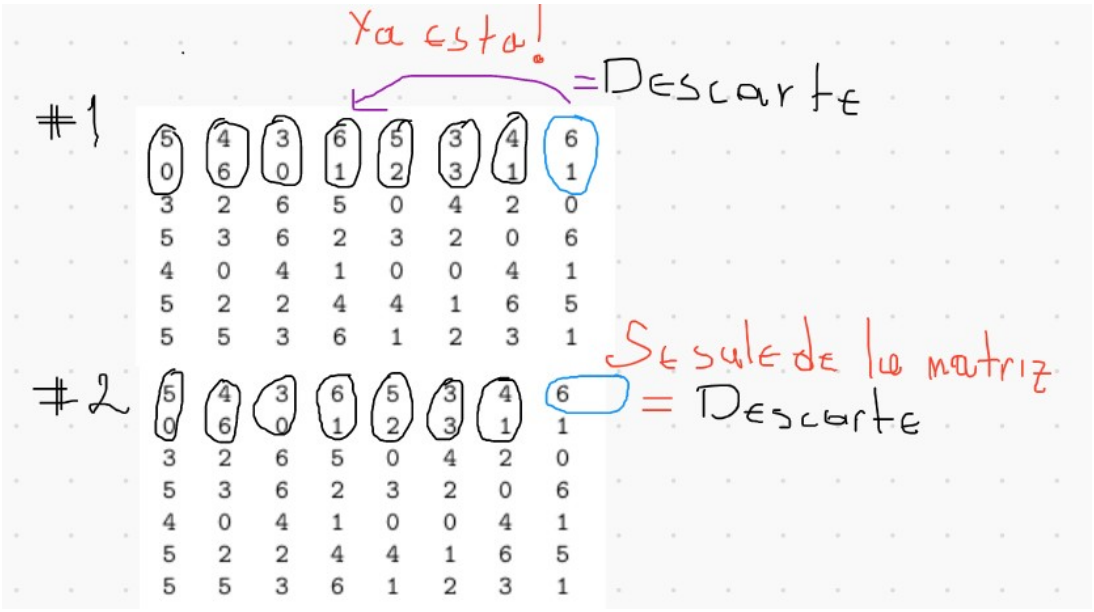
1. Encontrar pareja:
   - Primero debemos suponer que se inicia el recorrido en la posición (0,0) de la matriz **test,** allí tenemos dos opciones de elección de pareja, con el campo (0,1) o con el campo (1,0), es decir, (x+1,y) y (x,y+1), el algoritmo lo que hace es elegir las dos opciones, obviamente una primera que la otra.



   - Cuando se toma esos campos primero tenemos que validar dos cosas:
     - La primera cosa es que este campo no se salga del rango permitido, es decir, x no puede ser menor a 0 ni mayor a 6, y y no puede ser ni menor a 0 ni mayor a 7. También que el campo elegido no esté visitado, eso lo verificamos con la matriz **visitedFields.** Si el campo incumple con alguna de estas condiciones se descarta ese recorrido.
     - La segunda cosa a validar, que depende de que la anterior se halla cumplido, es validar que al formar la pareja el hueso resultante no este usado, para esto utilizamos la matriz

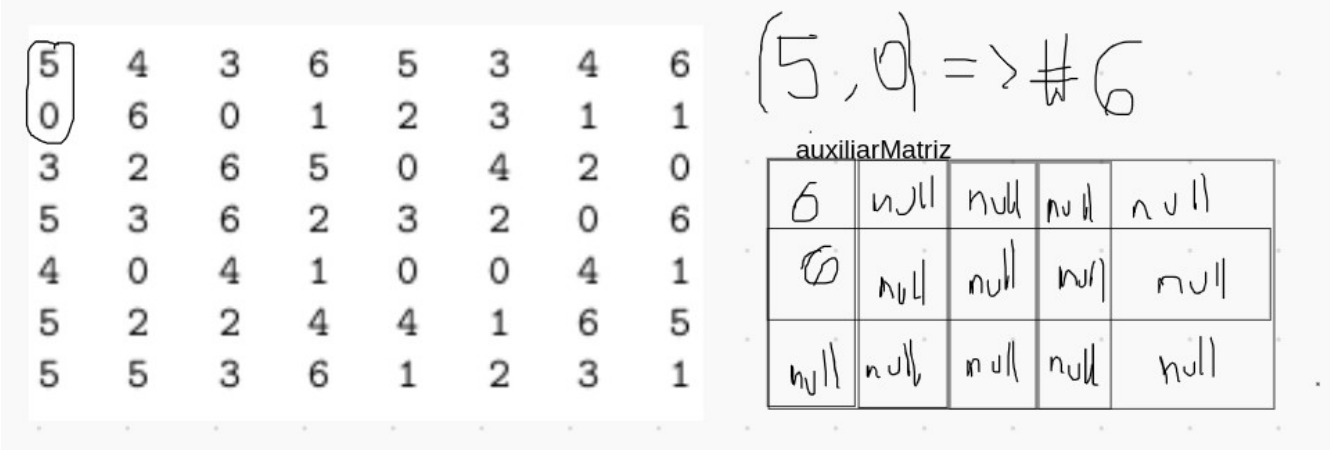
activatedBones pasandole el hueso correspondiente a la pareja. Dado el caso de que ya esté usado, descartamos esa ruta.

- Con los pasos anteriores ya hemos descartado muchas rutas que no llegaban a ningún lado. Ahora bien cuando todo salga bien procedemos con la siguiente parte.

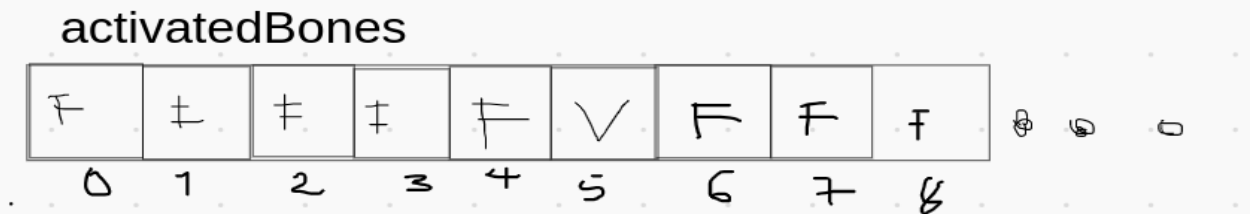


2. Poner valores y realizar recursión:

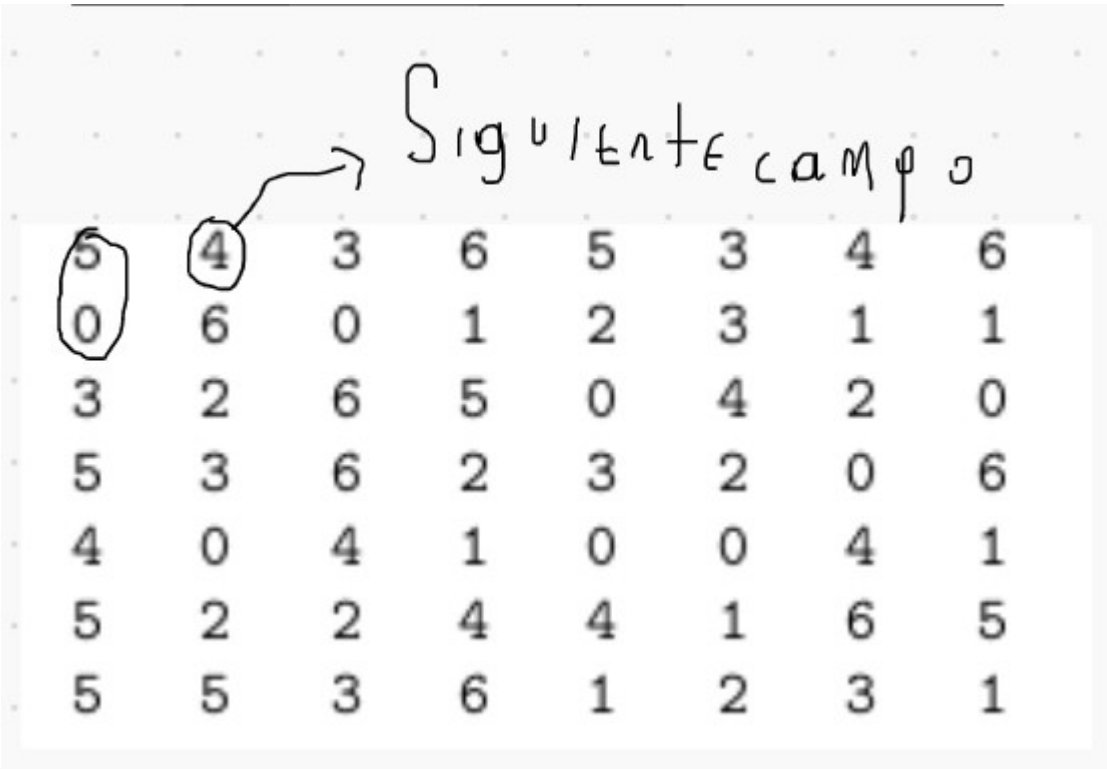
- Procedemos a poner en visitado los campos previamente validados, lo hacemos en la matriz **visitedFields,** también indicamos en la matriz auxiliarMatriz los huesos en ambos sentidos, es decir, tanto en el **row** y **col** actual como en la pareja encontrada.



- Luego de esto ponemos en **activatedBones** que el hueso ya está utilizado.



- Ahora buscamos cual será el siguiente campo a revisar para poder realizar la recursión, para esto recorremos la matriz **visitedFields** buscando el primer campo que no esté visitado obteniendo sus coordenadas en la matriz con ayuda del método **nextField.**



- Finalmente llamamos al método e ingresamos los mismos campos cambiando row y col que son elementos hallados en el paso anterior. Al llamar el método se sigue el mismo proceso recursivo de validación, búsqueda y recursión nuevamente. Ahora bien que pasa cuando en la ruta que se inicio desde cierto punto no lleva a ningún lado?

3. En respuesta del último punto de la parte 2, cuando pasa eso procedemos a quitar el visitado de los campos actuales, y también del hueso utilizado, consiguiendo así devolvernos ya que esa ruta no es la indicada. El factor que vuelve a “iniciar” la búsqueda es el método nextField ya que este se encarga de ir encontrando los primeros campos para la pareja, consiguiendo así un funcionamiento de backtraking. En las siguientes páginas se muestra un ejemplo del como va funcionando el backtraking con este algoritmo.
Tener en cuenta que en algunos casos se adelantaron pasos.

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

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

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

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

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

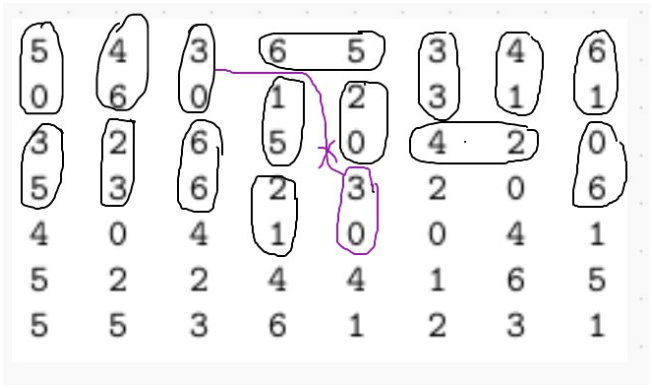
5	4	3	6	5	3	4	6
0	6	0	1	2	3	1	1
3	2	6	5	0	4	2	0
5	3	6	2	3	2	0	6
4	0	4	1	0	0	4	1
5	2	2	4	4	1	6	5
5	5	3	6	1	2	3	1

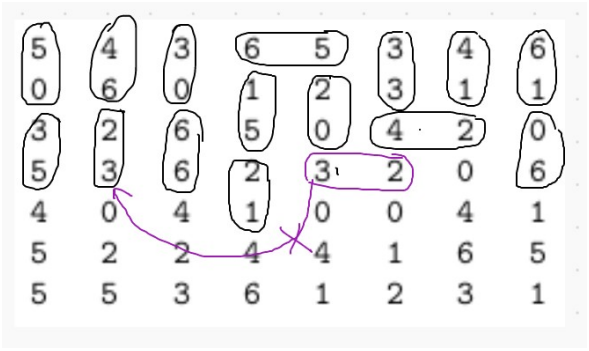
5	4	3	6	5	3	4	6
0	6	0	1	2	3	1	1
3	2	6	5	0	4	2	0
5	3	6	2	3	2	0	6
4	0	4	1	0	0	4	1
5	2	2	4	4	1	6	5
5	5	3	6	1	2	3	1

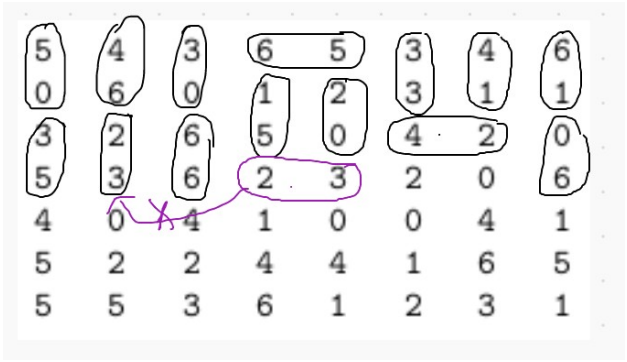
5	4	3	6	5	3	4	6
0	6	0	1	2	3	1	1
3	2	6	5	0	4	2	0
5	3	6	2	3	2	0	6
4	0	4	1	0	0	4	1
5	2	2	4	4	1	6	5
5	5	3	6	1	2	3	1

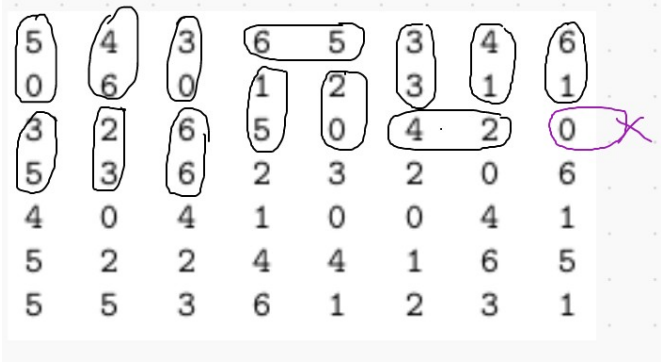
5	4	3	6	5	3	4	6
0	6	0	1	2	3	1	1
3	2	6	5	0	4	2	0
5	3	6	2	3	2	0	6
4	0	4	1	0	0	4	1
5	2	2	4	4	1	6	5
5	5	3	6	1	2	3	1

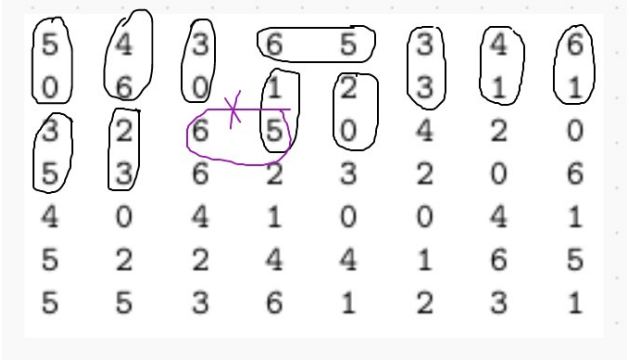
5	4	3	6	5	3	4	6
0	6	0	1	2	3	1	1
3	2	6	5	0	4	2	0
5	3	6	2	3	2	0	6
4	0	4	1	0	0	4	1
5	2	2	4	4	1	6	5
5	5	3	6	1	2	3	1

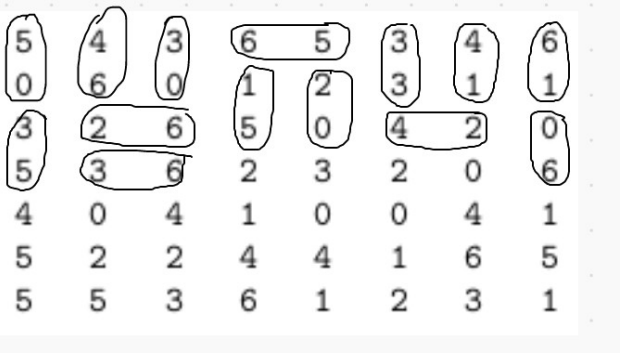






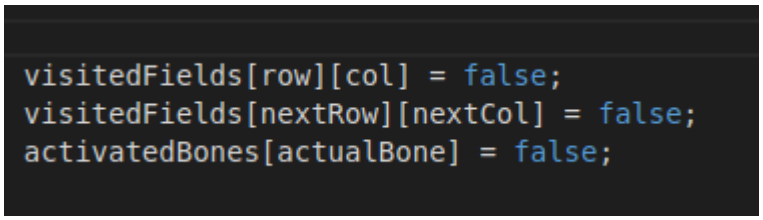






¿Cómo cambiaría el algoritmo si en lugar de preguntar por todas las soluciones preguntara por una solución?

Cómo la estructura del algoritmo es enfocada a encontrar todas las soluciones tocaría buscar la forma de parar la búsqueda, una solución podría ser poner una condición aquí:



que cuando una cierta variable que me guarde si hay o no solución sea verdadera no me ponga eso en falso sino que siga derecho, eso evitaría que se sigan buscando soluciones y solo tengamos la primera.