**COMPONENTES DEL EQUIPO –> B1\_06**

* Dídimo Javier Negro Castellanos
* Óscar Jiménez Jiménez
* Jorge Alberto Gómez León

**NOMBRE DEL REPOSITORIO -> B1\_06**

<https://github.com/didimoj/B1_06>

**CLASES CREADAS**

* **Main**

Clase principal de nuestro programa. En ella creamos una variable gestionArchivo y un Estado, que será el estado inicial (leído de nuestro fichero). Por último, creamos un Agente e invocamos al método getSolución pasándole como parámetro el estado inicial.

* **GestionArchivo**

Clase encargada de cargar, escribir e imprimir ficheros por pantalla.

En el método ***cargarArchivo*** leemos el fichero deseado y vamos guardando los datos en las variables x, y, k, max, c, f. después creamos un nuevo terreno, que será una matriz bidimensional con las filas y columnas indicadas en el fichero y le ponemos en cada posición la cantidad leída. Por último, comprobamos que las cantidades introducidas son correctas para realizar la distribución.

El método ***escribirArchivo*** se encarga de modificar un fichero, para ello usamos el BufferedWriter y en cada línea vamos añadiendo la información deseada.

El método ***imprimir*** mostrará la matriz solar con sus respectivos valores.

* **Estado**

Es la clase empleada para definir un estado. Está compuesta por las variables Terreno t, las posiciones X e Y donde se encuentra el tractor, el constructor, los métodos gets y sets y un método toString para mostrar información, donde se llama al método imprimir, que mostrará la matriz del terreno por pantalla.

* **Terreno**

Clase empleada para definir un terreno. Está formada por los atributos k (valor deseado en cada posición de la matriz), max (valor máximo que puede haber en cada posición de la matriz) y terreno (matriz bidimensional). Los métodos que usa son los gets necesarios para obtener los valores de los atributos.

* **GestionEstado**

Clase encarga de gestionar y realizar las operaciones necesarias de un estado. Está formada por un constructor y los métodos:

**- *crearEstado*,** al que le pasamos una lista de arrays, una acción y un estado y donde crearemos una copia del terreno, en la cual iremos realizando las modificaciones necesarias para la creación de nuevos estados. Por último, devolvemos el nuevo estado.

**- *distribucion*,** que le pasamos un estado como variable y creamos dos listas. Calculamos la cantidad de tierra sobrante en la posición donde se encuentra el tractor y la almacenamos en la variable cant. Llamamos al método back y almacenamos su valor en la lista todas1. Por último, creamos una lista de candidatos, que almacenará las acciones que podremos realizar en el estado actual y otra lista sucesores, donde se almacenarán los posibles estados sucesores del estado actual. Esta última lista será el valor de retorno del método.

**- *Distpos*,** al que le pasaremos como parámetros dos listas. Crearemos una lista de acciones, donde guardaremos cada una de las acciones posibles en el estado actual y que será lo que devuelve el método.

**- *esPosible*,** dados un array, una lista de arrays y un estado, comprueba si es posible mover una cantidad de tierra a una posición del terreno.

**- *crearLista*,** al que le pasamos un estado y creamos una nueva lista de arrays. En este método almacenamos en la lista los posibles movimientos que puede realizar el tractor, dependiendo de la posición del terreno en la que se encuentre.

**- *back***, método recursivo encargado de realizar el algoritmo Backtracking.

**- *suma*,** dado un array, un número de casillas y un número máximo, almacena en la variable suma la suma del valor de cada posición del array y devuelve si la suma total es igual al valor máximo o no.

**- *imprimir*,** pasándole un array bidimensional, se encarga de imprimirlo por pantalla.

* **Acciones**

Clase empleada para la definición de una acción. Contiene los atributos mov (vector con el movimiento que realiza) y dist (distribución), el constructor de la clase y los get y set necesarios, así como el método toString().

* **Distribucion**

Clase empleada para la definición de una distribución. Contiene los atributos posición (array) y cantidad, el constructor de la clase y los get y set necesarios, así como el método toString()

* **Agente**

Esta clase contiene un único atributo del tipo Frontera y está formado por el método constructor y los métodos:

**- *getSolucion*,** donde creamos una variable Random y un nuevo Nodo, que será el nodo raíz y lo añadimos a la frontera. Comprobamos que la frontera no está vacía y extraemos un nodo de ella. Comprobamos si el nodo es objetivo, en caso de que los sea, lo añadimos a una pila y vamos extrayendo todos los elementos de esa pila para encontrar la solución; en caso de que el nodo no sea objetivo, obtenemos sus nodos sucesores, que los introduciremos a la frontera.

**- *getId*,** que le pasamos un estado y en el que utilizamos la función hash, encriptada mediante MD5.

* **Frontera**

Clase que define como crear una frontera. Contiene el atributo lista, que es una cola de prioridad de nodos, el constructor y los métodos insertar nodo, eliminar nodo y comprobar si la frontera está vacía.

* **Nodo**

Clase que define cómo crear un nodo. Contiene los atributos id, estado, costo, valor y parent, el constructor de la clase, los get y set correspondientes y los métodos toString, equals (para comparar si dos nodos son iguales), comparteTo (compara si un nodo es mayor, menor o igual que otro) y getSucesores (devuelve los estados sucesores).

**ESTRUCTURAS Y GRÁFICOS DE LOS ARTEFACTOS CREADOS**

* **ArrayList<int[]> todas1** -> Una lista de vectores de enteros que contendrá todas las posibles distribuciones que puede realizar el tractor a las casillas vecinas, es decir, las posibilidades que tiene el tractor para mover la arena a las casillas vecinas. Ejemplo:

dist=[0, 0, 0, 3]

dist=[0, 0, 1, 2]

dist=[0, 2, 0, 1]

dist=[1, 1, 1, 0]

.

.

.

* **ArrayList<int[]> l** -> Lista de vectores de enteros que contiene los posibles movimientos, es decir, las posiciones (x, y) de los distintos vecinos con respecto a la casilla en la que se encuentra el tractor en su estado actual. Ejemplo:

mov=[2, 1]

mov=[1, 0]

mov=[0, 1]

.

.

.

* **int[][] terreno** -> Matriz de enteros que se asemeja al terreno en el que se encuentra el tractor. Los números en esta matriz terreno significan la cantidad de arena que hay en esa respectiva casilla. Ejemplo:

|  |  |  |
| --- | --- | --- |
| **3** | **5** | **2** |
| **5** | **8** | **4** |
| **4** | **3** | **8** |

* **ArrayList<Acciones> candidatos** -> Lista de acciones que contendrá las posibles acciones que podemos realizar desde el estado actual.
* **ArrayList<Estado> sucesores** -> Matriz de estados que contendrá los posibles estados a los que podemos llegar desde el estado actual.
* **Frontera f** -> Cola de prioridad donde almacenaremos los nodos que se encuentran en la frontera en cada momento.
* **Nodo nodo** -> Objeto referente a los nodos con los que iremos creando el árbol para buscar la solución. Almacenará el Id del nodo, el estado, el costo de realizar la acción, el valor del mismo y el identificador del nodo padre.

**------- ACCIONES POSIBLES --------**

Acciones [Movimiento=[1, 2], Distribucion=[0, 0, 0, 2]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[0, 0, 1, 1]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[0, 0, 2, 0]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[0, 1, 0, 1]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[0, 1, 1, 0]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[1, 0, 0, 1]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[1, 0, 1, 0]], Coste: 1

Acciones [Movimiento=[1, 2], Distribucion=[1, 1, 0, 0]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[0, 0, 0, 2]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[0, 0, 1, 1]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[0, 0, 2, 0]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[0, 1, 0, 1]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[0, 1, 1, 0]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[1, 0, 0, 1]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[1, 0, 1, 0]], Coste: 1

Acciones [Movimiento=[3, 2], Distribucion=[1, 1, 0, 0]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[0, 0, 0, 2]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[0, 0, 1, 1]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[0, 0, 2, 0]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[0, 1, 0, 1]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[0, 1, 1, 0]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[1, 0, 0, 1]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[1, 0, 1, 0]], Coste: 1

Acciones [Movimiento=[2, 1], Distribucion=[1, 1, 0, 0]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[0, 0, 0, 2]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[0, 0, 1, 1]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[0, 0, 2, 0]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[0, 1, 0, 1]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[0, 1, 1, 0]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[1, 0, 0, 1]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[1, 0, 1, 0]], Coste: 1

Acciones [Movimiento=[2, 3], Distribucion=[1, 1, 0, 0]], Coste: 1

**TOTAL ACCIONES = 32**