

PRACTICA INTELIGENTES

Carlos Coello Ruedas, Alejandro Medina Jiménez, Iván Illán Barraya

Practica 3

MAIN DE NUESTRO PROGRAMA.

Constructor and Description

PracticaInteligente()

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier and Type	Method and Description
static boolean	Busqueda_Acotada(Problema prob, practicaInteligente.tipoBusqueda estrategia, int Prof_Max, int opcion)
static boolean	Busqueda(Problema prob, practicaInteligente.tipoBusqueda estrategia, int Prof_Max, int Inc_Prof)
static java.util.ArrayList<Nodo>	CreaListaNodosArbol(java.util.ArrayList<Sucesor> LS, Nodo n_actual, int Prof_Max, practicaInteligente.tipoBusqueda estrategia)
static void	main(java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Nuestra clase principal se llama “PracticaInteligente” y en ella tenemos los métodos que se muestran en la imagen:

- Búsqueda Acotada: realiza a través del árbol que hemos generado con CreaListaNodosArbol una búsqueda acotada para resolver nuestro problema, si llega el nodo destino devuelve un valor de True
- CreaListaNodo creamos una lista con los nodos del árbol y según el tipo de búsqueda los nodos tendrán ciertas características. Por ejemplo, si se tratase de coste uniforme los nodos tendrían el coste actual + el coste del sucesor....
- Búsqueda: Es el que se encarga de decir si la búsqueda ha sido un éxito o no.

Luego el main se trata de un pequeño menú el cual nos da a elegir entre las estrategias disponibles para llevar a cabo nuestra búsqueda que son:

- Profundidad.
- Anchura.
- Coste uniforme.

-NUEVAS MODIFICACIONES.

Hemos creado una nueva opción para el A*, ahora a la búsqueda acotada le pasamos un numero que va a ser la opción seleccionada para que cuando la heurística sea 0 corte y no muestre más.

CLASE PROBLEMA

Constructor Summary

Constructors

Constructor and Description

`Problema(EspaciodeEstados espaciodeestados, Estado estadoinicial)`

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

`EspaciodeEstados`

`getEspaciodeestados()`

`Estado`

`getEstadoinicial()`

`boolean`

`Objetivo(Estado e)`

`void`

`setEspaciodeestados(EspaciodeEstados espaciodeestados)`

`void`

`setEstadoinicial(Estado estadoinicial)`

La clase Problema tendrá una instancia de la clase EspaciodeEstados y el estado inicial que puede ser dado a través de un fichero, de lectura por teclado o random. También hemos incluido los Getters y Setters para obtener este espacio de estados y este estado inicial. También tenemos un método objetivo para saber si este estado se trata del estado al que queremos llegar, es decir al objetivo. El método donde se lleva a cabo esta comprobación se encuentra en la clase Estado.

Clase Estado

Constructor and Description

```
Estado(Casilla tractor, int k, int max, int filas, int columnas)
```

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	backrepartir(int etapa, int s, Casilla[] adyacentes, int[] sol, java.util.ArrayList soluciones)
int	cantidadPosible(Casilla c)
Casilla[]	casillasAdyacentes()
int	Costo(accion a)
boolean	esPosible(int i, int etapa, int[] sol, int s)
boolean	esSolucion(int[] sol, Casilla[] adyacentes, int s)
boolean	estaDentro(Casilla c)
java.util.ArrayList<accion>	getAcciones()
Casilla	getCasilla(Casilla c)
Casilla[][]	getCasillas()
int	getColumnas()
Estado	getEstado(accion a)
int	getFilas()
int	getK()
int	getMax()
int	gets()
Casilla	getTractor()
void	iniciarTerreno(int[] v)
void	setCasilla(Casilla c)
boolean	testObjetivo()
java.lang.String	toString()

En esta clase tenemos hecho un backtracking que es el método Backrepartir gracias a el llevamos a cabo la repartición del terreno, este método llama a esPosible y esSolucion para comprobar que se puede introducir esa cantidad en esa casilla o para saber si es una solución a lo que hemos llegado o no .

Esta clase estado tendrá como parámetros una casilla que corresponderá con el tractor la cantidad a repartir, la cantidad máxima las filas y columnas del terreno.

Hemos añadido lo que nos pedían en esta práctica que era añadir un método Costo dentro de la clase estado que tenga como argumento una acción y devuelva el costo de la acción.

El costo de una acción es el total de arena que se va a mover con la acción más 1.

Aquí es donde se encuentra nuestro método test Objetivo nombrado antes para saber si hemos llegado al objetivo.

-NUEVAS MODIFICACIONES.

Hemos implementado el método getCasillas() para que nos devuelva el tablero, con el numero que contiene cada casilla.

Clase Casilla

Constructors

Constructor and Description

Casilla(int fila, int columna)

Casilla(int fila, int columna, int cantidad)

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

int getCantidad()

int getColumna()

int getFila()

void setCantidad(int cantidad)

void setColumna(int columna)

void setFila(int fila)

java.lang.String toString()

Esta sería nuestra clase Casilla en la cual tendríamos la cantidad de la casilla, con su fila y columna correspondiente.
También tenemos otro constructor con solo la fila y la columna para crear el tractor.
Hemos generado sus correspondientes getter y setter para cambiar los atributos nombrados anteriormente y poder obtenerlos.

Clase Sucesor

Constructors
Constructor and Description
Sucesor(accion a, Estado e, int coste)

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description
	accion	getA()
	int	getCoste()
	Estado	getE()
	void	setA(accion a)
	void	setCoste(int coste)
	void	setE(Estado e)
	java.lang.String	toString()

En esta clase creamos los sucesores correspondientes con su acción, coste y el Estado que le corresponde.

Clase Frontera

Constructors

Constructor and Description
Frontera()

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description
	void	CrearFrontera()
	Nodo	Elimina()
	boolean	EsVacia()
	void	Insertar(Nodo e)

Aquí tenemos nuestra clase frontera la cual será una priority queue en la cual almacenaremos simplemente los nodos de la frontera de nuestro árbol, y tendremos los métodos de eliminar, insertar un nodo en la frontera y comprobar si está vacía.

Clase EspacioDeEstados

Constructors	
Constructor and Description	
EspaciodeEstados()	

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description
java.util.ArrayList<Sucesor>		sucesor(Estado e)

Esta es nuestra clase espacio de estados en la cual vamos a crear una arraylist con los sucesores correspondientes, y esta instancia se usará en la clase Problema. Para las distintas combinaciones a distribuir según la opción.

Clase Nodo

Nodo(Nodo padre, Estado e, accion a, int costo, int profundidad, int valor)	
---	--

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description
int		compareTo(Nodo e)
accion		getA()
int		getCosto()
Estado		getE()
int		getHeuristica(Estado e)
Nodo		getPadre()
int		getProfundidad()
int		getValor()
void		setA(accion a)
void		setCosto(int costo)
void		setE(Estado e)
void		setPadre(Nodo padre)
void		setProfundidad(int profundidad)
void		setValor(int valor)
java.lang.String		toString()

-NUEVAS MODIFICACIONES.

En la clase nodo hemos implementado el método getHeuristica() que calcula la heurística al que le pasamos como parámetro el estado del nodo, la heurística es el número de casillas que no tienen asignada la cantidad k de arena, es decir, la cantidad correcta.

