

Ejemplo de resolución de un problema con AIMA

● **Problema de los misioneros y los caníbales**

- Representación del problema
- Ejecución de la búsqueda

Ejemplo: los misioneros y los caníbales

Representación formal del problema

- Estado inicial: **(3, 3, 1)**. Estado objetivo: **(0, 0, 0)**
- Restricciones: estados no válidos (condición de peligro **(NM, NC, B)**)
 - **$(NM < NC \wedge NM \neq 0) \vee (NM > NC \wedge NM \neq 3)$**
- Operadores: **cruzaM, cruzaMM, cruzaC, cruzaCC, cruzaMC**
 - Ejemplo de especificación para **cruzaM (NM, NC, B)**
 - Precondiciones
 - **$(B = 1 \wedge NM > 0) \vee (B = 0 \wedge NM < 3)$**
 - Postcondiciones
 - **$(NM, NC, 1) \rightarrow (NM - 1, NC, 0)$**
 - **$(NM, NC, 0) \rightarrow (NM + 1, NC, 1)$**
 - **Estado destino no peligroso**
 - Coste del operador: 1
 - Especificación para **cruzaMM, cruzaC,...**
- Coste de la solución = número de operadores aplicados

Representación de un problema concreto en AIMA: Misioneros

Necesitamos 5 clases:

- 1) Estado del problema → **EstadoMisioneros**
- 2) Test de estado objetivo → **MisionerosGoalTest**
- 3) Acciones posibles → **MisionerosActionsFunction**
- 4) Resultado de cada acción → **MisionerosResultFunction**
- 5) Función de coste → **MisionerosStepCostFunction**

1) Estado del problema → EstadoMisioneros

Clase independiente del Framework → EstadoMisioneros.java

- Se define la **representación** de cada estado:

un tablero, un mapa, varios valores, etc...

- `private int numMisioneros; // numMisioneros en orilla izquierda`
- `private int numCanibales; // numCanibales en orilla izquierda`
- `private boolean barcaIzq; // true si barca en orilla izquierda`

- y las acciones posibles (actions) como objetos DynamicAction

- `public static Action M = new DynamicAction("M");`
- `public static Action MM = new DynamicAction("MM");`
- `public static Action C = new DynamicAction("C");`
- `public static Action CC = new DynamicAction("CC");`
- `public static Action MC = new DynamicAction("MC");`

1) Estado del problema → EstadoMisioneros

- Su constructor sirve para generar estados → `public EstadoMisioneros()`
En particular se tiene que usar para generar el estado inicial

```
public EstadoMisioneros() {  
    this(3, 3, true);  
}  
  
public EstadoMisioneros(EstadoMisioneros mc) {  
    this(mc.getnMisioneros(), mc.getnCanibales(), mc.isBarcaIzq());  
}  
  
public EstadoMisioneros(int nMisioneros, int nCanibales, boolean  
barcaIzq) {  
    this.nMisioneros = nMisioneros;  
    this.nCanibales = nCanibales;  
    this.barcaIzq = barcaIzq;  
}
```

1) Estado del problema → EstadoMisioneros

- Incluye
 - los operadores para transformar un estado en otro válido
 - las precondiciones y poscondiciones
 - La comprobación de estados no válidos (estados de peligro)

1) Estado del problema → EstadoMisioneros

- Incluye los operadores para transformar un estado en otro estado

```
public void moveM() {  
    if (barcaIzq)  
        nMisioneros--;  
    else  
        nMisioneros++;  
    cambiarDeOrilla();  
}
```

1) Estado del problema → EstadoMisioneros

- Incluye métodos que comprueban si un operador se puede aplicar teniendo en cuenta las precondiciones, las poscondiciones y los estados de peligro

```
public boolean movimientoValido(Action where) {
    boolean valido = false;
    if (where.equals(M)) {
        if (((barcaIzq && nMisioneros > 0) || (!barcaIzq && nMisioneros < 3)) {
            EstadoMisioneros estadoSiguiente = new EstadoMisioneros(this);
            estadoSiguiente.moveM();
            valido = !estadoSiguiente.peligro();
        }
        else valido = false;
    }
    else if (where.equals(MM)) {
        ...

    return valido;
}
```


1) Estado del problema → EstadoMisioneros

- Incluye comprobación de estados no válidos (estados de peligro)

```
private boolean peligro() {  
    if (((nMisioneros < nCanibales) && (nMisioneros != 0))  
        || ((nMisioneros > nCanibales) && (nMisioneros != 3)))  
        return true;  
    else  
        return false;  
}
```

1) Estado del problema → EstadoMisioneros

- redefinir hashCode() y equals()

```

public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if ((o == null) || (this.getClass() != o.getClass())) {
        return false;
    }
    EstadoMisioneros otroEstado = (EstadoMisioneros) o;
    if ((this.nMisioneros == otroEstado.getnMisioneros()) &&
        (this.nCanibales == otroEstado.getnCanibales()) &&
        (this.barcaIzq == otroEstado.isBarcaIzq()))
        return true;
    else
        return false;
}

public int hashCode() {
    return (100 * nMisioneros) + (10 * nCanibales) + (barcaIzq ? 1 : 0);
}

```

2) Test de estado objetivo → `MisionerosGoalTest`

- Test para comprobar si el estado actual es un estado objetivo

Clase que implementa la interfaz **GoalTest** - > **MisionerosGoalTest.java**

Hay que implementar el método **public boolean isGoalState(Object state)** que determina si un estado es estado final

```
public class MisionerosGoalTest implements GoalTest {  
    EstadoMisioneros goal = new EstadoMisioneros(0, 0, false);  
    public boolean isGoalState(Object state) {  
        EstadoMisioneros estado = (EstadoMisioneros) state;  
        return estado.equals(goal);  
    }  
}
```

3) Acciones posibles → `MisionerosActionsFunction`

- Obtener el conjunto de acciones que se pueden ejecutar en un determinado estado (`ACTIONS(s)`)

Clase privada estática que implementa la interfaz `ActionsFunction` -> `MisionerosActionsFunction`

Hay que implementar el método **`public Set<Action> actions(Object state)`**

- que devuelve una lista con las acciones posibles

```
private static class MisionerosActionsFunction implements ActionsFunction
{
    public Set<Action> actions(Object state)
    {
        EstadoMisioneros estado = (EstadoMisioneros) state;
        // lista de acciones posibles
        Set<Action> actions = new LinkedHashSet<Action>();
        // si se cumplen las precondiciones y no se va a un estado de peligro entonces
        // se añade la acción a la lista de acciones posibles
        if (estado.movimientoValido(EstadoMisioneros.M))
            actions.add(EstadoMisioneros.M);
        if ....

        return actions;
    }
}
```

4) Resultado de cada acción → `MisionerosResultFunction`

- Definir el cambio de estado que se produce al ejecutar una acción sobre un estado (`RESULT(s,a)`)

Clase privada estática que implementa la interfaz `ResultFunction => MisionerosResultFunction`

Hay que implementar el método `public Object result(Object s, Action a)`

- que devuelve el estado que resulta de aplicar la acción a al estado s

```
private static class MisionerosResultFunction implements ResultFunction
{
    public Object result(Object s, Action a) {
        EstadoMisioneros estado = (EstadoMisioneros) s;

        if (EstadoMisioneros.M.equals(a)) {
            EstadoMisioneros nuevoEstado = new EstadoMisioneros(estado);
            nuevoEstado.moveM();
            return nuevoEstado;
        }
        elseif
        ...
        // The Action is not understood or is a NoOp
        // the result will be the current state.
        return s;
    }
}
```

Clase Factoría

- 3) y 4) se definen en una clase factoría (MisionerosFunctionFactory.java) con dos objetos estáticos ActionsFunction y ResultFunction devueltos por sendos métodos estáticos public static ActionsFunction getActionsFunction() y public static ResultFunction getResultFunction()

```
public class MisionerosFunctionFactory
{
    private static ActionsFunction _actionsFunction = null;
    private static ResultFunction _resultFunction = null;

    public static ActionsFunction getActionsFunction() {
        if (null == _actionsFunction) {
            _actionsFunction = new MisionerosActionsFunction();
        }
        return _actionsFunction;
    }

    public static ResultFunction getResultFunction() {
        if (null == _resultFunction) {
            _resultFunction = new MisionerosResultFunction();
        }
        return _resultFunction;
    }
}
```

5) Función de coste → `MisionerosStepCostFunction`

– Definir una función de coste

Clase que implementa la interfaz **StepCostFunction**

Hay que implementar el método **public double c(Object from, Action a, Object to)**

- que determina el coste de pasar del estado **from** al estado **to** con la acción **a**

```
public class MisionerosStepCostFunction implements StepCostFunction
    public double c(Object stateFrom, Action action, Object stateTo)
        return 1;
```

- ❑ Es opcional, si no se crea la clase se usa la función de coste por defecto (`DefaultStepCostFunction`) que asigna 1 como coste de una acción

```
public class DefaultStepCostFunction implements StepCostFunction
    public double c(Object stateFrom, Action action, Object stateTo)
        return 1;
```

Ejecutar una Demo de Búsqueda (MisionerosDemo.java)

```

public class MisionerosDemo {
    static EstadoMisioneros estadoInicial = new EstadoMisioneros();
    public static void main(String[] args) {
        MisionerosBFSDemo();
    }
    private static void MisionerosBFSDemo()
        System.out.println("\nMisionerosBFSDemo-->");
        try
            // Crear un objeto Problem con la representación de estados y operadores
            Problem problem = new Problem(estadoInicial,
            MisionerosFunctionFactory.getActionsFunction(), MisionerosFunctionFactory.getResultFunction(),
            new MisionerosGoalTest()); // si no hay función de coste en el constructor se usa el coste por defecto
            // si hay función de coste hay que añadir el objeto correspondiente: new MisionerosStepCostFunction()

            // indicar el tipo de búsqueda
            Search search = new BreadthFirstSearch(); // búsqueda en anchura

            // crear un agente que realice la búsqueda sobre el problema
            SearchAgent agent = new SearchAgent(problem, search);

            // escribir la solución encontrada (operadores aplicados) e información sobre los recursos utilizados
            printActions(agent.getActions());
            printInstrumentation(agent.getInstrumentation());
            // Hay que implementar estos métodos en la clase Demo (copiar de aim.gui.demo.search.EightPuzzleDemo)

        catch (Exception e)
            e.printStackTrace();

```